National Institute of
Standards and Technology
U.S. Department of Commerce

Internal Group Report

Feb 12, 2013

SAJACC Working Group Recommendations to NIST

National Institute of Standards and Technology

NIST Cloud Computing

Standards Acceleration to Jumpstart Adoption
of Cloud Computing (SAJACC) Working Group

Phase I group report and recommendations

as presented at and incorporating input from the

NIST Joint Cloud Computing and
Big Data Workshop & Forum

January 15-17, 2013

Executive Summary and Recommendations

The NIST Cloud Computing "Standards Acceleration to Jumpstart Adoption of Cloud Computing" (SAJACC) Working Group pursued a strategic process to facilitate development of testing methodologies applicable to cloud computing products and standards. The purpose of this process was to create formal US Government (USG)-based use cases and validation mechanisms that would ensure identification of the detailed capabilities of cloud computing products and standards in terms of their ability to support the US Government's "Cloud First" information technology strategy, and to test cloud computing products and standards against these use cases.

The SAJAAC process was designed to incorporate the output of other NIST Cloud Computing working groups where possible, especially that of the Business Use Case Working Group, and to identify specific features that reflect USG priorities for cloud computing, which include aspects related to security, interoperability and portability. Future work is anticipated to extend the SAJACC framework to features that touch on other recently identified priorities, including aspects of accessibility and performance. By constantly integrating such aspects identified by other related NIST Cloud Computing topical work, the NIST SAJAAC process is designed to produce a validation process that will help support a sustainable, secure USG cloud infrastructure.

This working group report captures the results of this process to date and makes the following conclusions and recommendations to NIST to proceed from Phase I to future Phase II work of the SAJACC group:

1. Replace the SAJACC use case internal organization with one based on the current structure of the NIST Cloud Computing Reference Architecture and Taxonomy;
2. Add further use cases based on current extensions to this taxonomy for recently developed Cloud SLA Metrics and NIST Cloud Computing Security components;
3. Integrate further input as necessary from the NIST Business Use Case and Standards Roadmap groups, and work closely with these groups to identify additional use cases;
4. Study and adopt use case template elements from the US VA Bronze, Silver and Gold Use Cases and from additional formal input from US Government agencies;
5. Add automation and tooling, if possible, to the NIST web site to support community downloading of the NIST SAJACC use cases and their associated templates for testing scenarios and uploading of externally produced test results;
6. Conduct, invite and document additional use case demonstrations of cloud standards and applicable products against the SAJACC use cases to illustrate their features;
7. Solicit and add further recommendations from the community at large through meetings of the SAJACC working group.

This report therefore comprises the conclusion of Phase I of the SAJACC process to date, and the plan for initiation of Phase II with the goal to implement the above recommendations.

Acknowledgements

This report captures the results of a community-based input process that began at the May 11, 2010 Gaithersburg Use Case workshop led by NIST staff member Lee Badger, and continued by participants in that workshop and by subsequent SAJACC WG members, including Lee Badger, Jin Tong, Babak Jahromi, Hemma Prafullchandra, Alan Sill and Gregg Brown over the course of the 2010-2011 time period to produce the first online SAJACC Use Case list. The current document reflects ongoing contributions and discussions by the membership of the NIST Cloud Computing SAJACC Working Group, chaired during the 2012 time frame by Alan Sill of Texas Tech University and Eugene Luster of R2AD Corporation.

The document also contains material gathered from reports and group output from the NIST Cloud Computing Reference Architecture and Taxonomy Working Group led by Robert Bohn and John Messina, the NIST Cloud Computing Standards Roadmap group chaired by Michael Hogan and Annie Sokol, the NIST Cloud Computing Reference Architecture and Taxonomy Cloud Service Level Agreement subgroup chaired by Frederic Devaulx and the NIST Cloud Computing Security group chaired by Michaela Iorga and Lee Badger. Such input from these latter groups is largely in the form of transcriptions of the mind maps and organizational structures of the output documents and public meeting notes of those groups; any errors in transcription of these results should be blamed on the current authors of this report and are not intended to reflect the complete output of those groups.

This report has been authored by the current NIST Cloud Computing SAJACC Working Group chairs, Eugene Luster and Alan Sill, based on the entire collected SAJACC community TWiki contents and meeting notes, with substantial input from the current working group members. It represents a transcription of the current SAJACC Use Cases along with suggestions for future reconfiguration. Specific acknowledgement is made of valuable input provided the NIST SAJACC Working Group during this process, including but not limited to the following members (listed in alphabetical order):

Lee Badger (NIST); Brent Bachorik (Fusion PPT); Dick Brackney (Microsoft), John Calhoon (Microsoft); Beth Chinn (IITA/BICES); Yuri Demchenko (UvA, NL); Dave Harper (JHU/APL); Jenny Huang (AT&T); A. Larry Gurule (Computer Sciences Corporation); Molly Johnson (US Department of Navy); Nancy Landreville (USVA); Eugene Luster (R2AD Corporation LLC); Shamun Mahmud (DLT Solutions); Tom Mancuso (SNIA), Gary Mazzaferro (AlloyCloud Inc.), Steven McGee (SAW Concepts); Linda Pelekoudas (SDS); Frank Peluso (Centuric); Rod Peterson (USVA); Gilbert Pilz (Oracle); Anh-Hong Rucker (NASA JPL); Alan Sill (Texas Tech University), Eric Simmon (NIST); Annie Sokol (NIST); Ken Stavinoha (Cisco); Steve Woodward (Woodward Systems).

Table of Contents

1. Introduction

Even though the origins of cloud computing (i.e., utility computing) date from the 1960s, cloud computing in 2013 is still a youthful field. Accumulated advances in network capacity, nearly ubiquitous connectivity, storage capacity, and efficient hardware virtualization have created new business opportunities: cloud providers can now rent computing resources over network connections to customers at costs that are very competitive with direct customer ownership. Moreover, cloud-based resource rentals have the potential to be far more flexible and convenient than resource ownership for end users. To maximize these opportunities, cloud providers are inventing new tools to manage remote, network-accessed, rental computing resources. These tools range from low-level system management utilities to new programming languages, new middleware software stacks, and new scalable algorithms for computing on large collections of data under the assumption that large numbers of compute and storage servers can be rapidly provisioned under software control.

The US Government, along with other potential cloud-computing customers, has a strong interest in the evolution of a vibrant and competitive cloud-computing marketplace. Prospective customers, however, legitimately seek answers to three important questions:

1. If a customer wishes to move their workload away from a cloud provider, can that be done at low cost and disruption? I.e., does the cloud provide portability?
2. Can a customer concurrently employ multiple cloud providers to achieve a single goal at low cost? I.e., does the cloud provide interoperability?
3. What support for security can cloud providers offer to allay concerns about how customer data is protected from unauthorized disclosure or modification; and what kinds of availability requirements can cloud providers satisfy? I.e., does the cloud provide support for security?

Generating high-quality cloud computing standards is one way to help to answer these questions, but standards development is time-consuming. In the absence of existing standards, there is a risk that short-term industry decisions affecting these questions, if not carefully considered in the short term, may become legacy constraints.

The Standards Acceleration to Jumpstart Adoption of Cloud Computing (SAJACC) project at the National Institute of Standards and Technology (NIST) seeks to generate concrete data about how different kinds of cloud system interfaces can support portability, interoperability, and security. By showing worked examples, the SAJACC project seeks to facilitate Standards Development Organizations in their efforts to develop high-quality standards that address these important needs.

The operation of SAJACC is conceptually simple. SAJACC will iteratively:

1. Develop a set of cloud system use cases that express selected portability, interoperability, and security concerns that cloud users may have;
2. Select a small set of existing cloud system interfaces that can be used for testing purposes;
3. Develop a test driver, for each use case and selected system interface, that represents (to the extent possible) the operation of the use case on the selected system interface;

4. Run the test drivers and document the extent each test driver can run on each selected system interface, and document any portability, interoperability, or security implications of the test run; and
5. Publish all use cases, test codes, and test results on the openly-accessible NIST Cloud Portal (www.nist.gov/itl/cloud), for use by Standards Development Organizations and other interested parties.

Figure 1 depicts the NIST Cloud Portal (currently in the form of a community TWiki web site) and the main steps of starting the SAJACC process. At this time of this writing, SAJACC has completed several iterations of steps 1-3, and exploratory activities conducted in a community setting for the process described in step 4 of the figure, resulting in output that has been documented in the NIST Cloud Standards TWiki. The working group has thus demonstrated but has NOT yet carried out the complete program of cloud standards validation described in this figure.

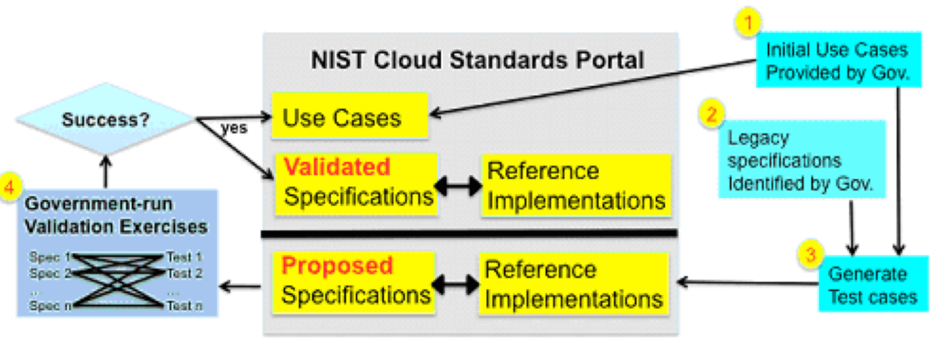

NIST Cloud Standards Portal
1
Initial Use Cases Provided by Gov.
Success?
yes
Use Cases
2
Legacy specifications Identified by Gov.
Validated Specifications
Reference Implementations
4
Government-run Validation Exercises
Spec 1
Spec 2
Spec n
Test 1
Test 2
Test n
Proposed Specifications
Reference Implementations
3
Generate Test cases

Figure 1

The rest of this document describes the work done by the SAJACC working group so far, which has resulted in a set of preliminary use cases developed for the first pass through the SAJACC process and a set of initial demonstration validation evaluations. Through a series of open workshops, and through public comment and feedback, NIST will continue to refine these use cases and add new use cases as appropriate.

2. Terminology and Structure of the SAJACC Use Cases

Use cases are a well-known tool for expressing requirements at a high level. In this document we briefly and informally describe a set of use cases for cloud systems. We use the following informal definition of a use case.

Use Case: a description of how groups of users and their resources may interact with one or more cloud computing systems to achieve specific goals. This document adapts the informal use case template from [Cockburn]

The following sections present informal descriptions that focus on: actors and goals, success scenarios, failure conditions, and failure handling (Cockburn's terminology).

2.1 Important Actors for Public Clouds

Table 1 lists the actors that show up in the use cases. Importantly, the actors are disjoint and do not (currently) inherit from one another. We adopt the definition of "actor" given by [Cockburn] to be, essentially, anything with "behavior" such as a person or a program.

In our uses cases, we list the actors that participate. In some cases, an "entity" transforms itself from one kind of actor to another by, for example, authenticating itself (becoming an authenticated user). In this situation, the use case lists all the actors that appear at any point in

Actor Name	Description
unidentified-user	An entity in the Internet (human or script) that interacts with a cloud over the network and that has not been authenticated.
cloud-subscriber	A person or organization that has been authenticated to a cloud and maintains a business relationship with a cloud.
cloud-subscriber-user	A user of a cloud-subscriber organization who will be consuming the cloud service provided by the cloud-provider as an end user. For example, an organization's email user who is using a SaaS email service the organization subscribes to would be a cloud-subscriber's user.
cloud-subscriber-administrator	An administrator type of user of a cloud-subscriber organization that performs (cloud) system related administration tasks for the cloud-subscriber organization.
cloud-user	A person who is authenticated to a **cloud-provider** but does not have a financial relationship with the **cloud-provider**.
payment-broker	A financial institution that can charge a **cloud-subscriber** for cloud services, either by checking or credit card.
cloud-provider	An organization providing network services and charging **cloud-subscriber**s**.** A (public) **cloud-provider** provides services over the Internet.
transport-agent	A business organization that provides physical transport of storage media such as high-capacity hard drives.
legal-representative	A court, government investigator, or police.
identity-provider	An entity that is responsible for establishing and maintaining the digital identity associated with a person, organization, or (in some cases) a software program. [NSTIC]
attribute-authority	An entity that is responsible for creating and managing attributes (e.g., age, height) about digital identities, and for asserting facts about attribute values regarding an identity in response to requests. [NSTIC]
cloud-management-broker	A service providing cloud management capabilities over and above those of the **cloud-provider** and/or across multiple **cloud-provider**s. Service may be implemented as a commercial service apart from any **cloud-provider**, as cross-provider capabilities supplied by a cloud-provider or as **cloud-subscriber**-implemented management capabilities or tools

Table 1: Actors

time, and the scenarios of the use case document the points in which actor transitions occur.

Note: We are aware that there are some conflicts between this list of actors defined here and the taxonomy given in NIST Special Publication 500-293, US Government Cloud Computing Technology Roadmap, Release 1.0 (Draft), Volume II Useful Information for Cloud Adopters, which defines five major actors: cloud consumer, cloud provider, cloud carrier, cloud auditor, and cloud broker. These differences are not important for the purposes of the use case scenarios described in this current report, for which the definitions used in Table 1 have been used on a consistent basis. A later section of this report addresses suggested work to bring the SAJACC use case definitions, actors and scenarios into a closer match to the terminology used in the NIST published reports.

The following sections organize the use cases by whether they are about the general management, the interoperability between clouds, and by security issues. We believe that the use cases presented in sections 3, 4, and 5 are likely candidates for testing in the SAJACC project. The use cases presented in section 6, "Future Use Case Candidates" will be considered after the earlier use cases have been implemented. Additional discussion on the organization of these use cases with respect to output of other NIST Cloud Computing working groups is given in Section 8.

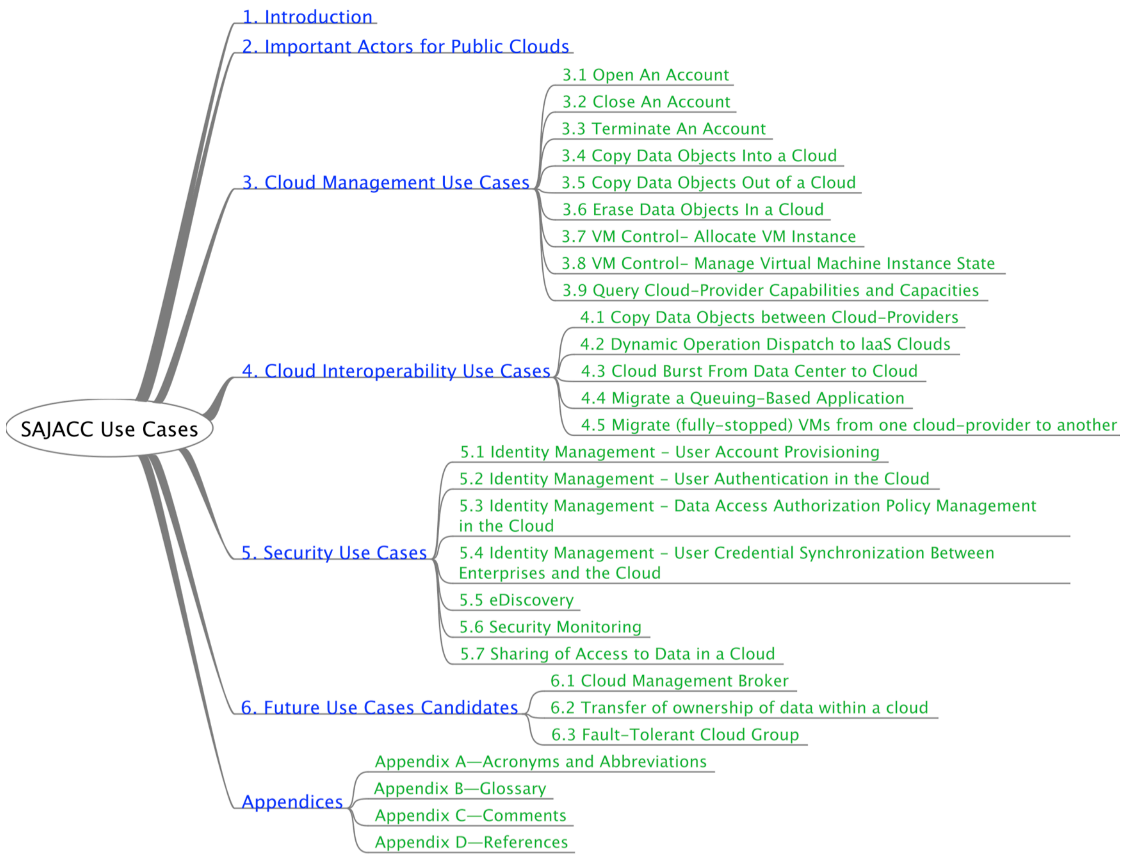

SAJACC Use Cases
1. Introduction
2. Important Actors for Public Clouds
3. Cloud Management Use Cases
3.1 Open An Account
3.2 Close An Account
3.3 Terminate An Account
3.4 Copy Data Objects Into a Cloud
3.5 Copy Data Objects Out of a Cloud
3.6 Erase Data Objects In a Cloud
3.7 VM Control- Allocate VM Instance
3.8 VM Control- Manage Virtual Machine Instance State
3.9 Query Cloud-Provider Capabilities and Capacities
4. Cloud Interoperability Use Cases
4.1 Copy Data Objects between Cloud-Providers
4.2 Dynamic Operation Dispatch to IaaS Clouds
4.3 Cloud Burst From Data Center to Cloud
4.4 Migrate a Queuing-Based Application
4.5 Migrate (fully-stopped) VMs from one cloud-provider to another
5. Security Use Cases
5.1 Identity Management - User Account Provisioning
5.2 Identity Management - User Authentication in the Cloud
5.3 Identity Management - Data Access Authorization Policy Management in the Cloud
5.4 Identity Management - User Credential Synchronization Between Enterprises and the Cloud
5.5 eDiscovery
5.6 Security Monitoring
5.7 Sharing of Access to Data in a Cloud
6. Future Use Cases Candidates
6.1 Cloud Management Broker
6.2 Transfer of ownership of data within a cloud
6.3 Fault-Tolerant Cloud Group
Appendices
Appendix A—Acronyms and Abbreviations
Appendix B—Glossary
Appendix C—Comments
Appendix D—References

Figure 2: SAJACC Use Case Organization

2.2 General Organization of the SAJACC Use Cases

Figure 2 provides an overall depiction of the SAJACC Use Case scenarios organized as a mind map. As noted above, this organization developed from early work in preparation of the SAJACC use cases. To avoid confusion, the numbering system for these use cases was kept consistent during the meeting series that followed. This procedure allowed reference to the use case scenarios numerically during the evaluation of the validation process for tests and demonstrations conducted against these scenarios.

3. Cloud Management Use Cases

The following use cases have been prepared to address scenarios related to cloud management operations. At the present time, they represent a substantial sampling of typical cloud management use cases, but do not yet comprise a complete or comprehensive set. Work is ongoing within the SAJACC group to identify a structure that can provide a more comprehensive set of operations based on community input, existing standards and cloud product application programmer interface (API) capabilities, the NIST Cloud Computing Reference Architecture and existing best practices for cloud management.

3.1 Open An Account

Actors: **unidentified-user**, **cloud-subscriber**, **payment-broker**, **cloud-provider**.

Goals: **Cloud-provider** opens a new account for an **unidentified-user** who then becomes a **cloud-subscriber**.

Assumptions: A **cloud-provider** 's account creation web page describes the service offered and the payment mechanisms. An **unidentified-user** can access the **cloud-provider** 's account creation web page.

Success Scenario: **(open, IaaS, PaaS, SaaS):** An **unidentified-user** accesses a **cloud-provider** 's account creation web page. The **unidentified-user** provides: (1) a unique name for the new account; (2) information about the **unidentified-user** 's financial information; and (3) when the **unidentified-user** wants the account opened. The **cloud-provider** verifies the **unidentified-user** 's financial information; if the information is deemed valid by **cloud-provider**, the **unidentified-user** becomes a **cloud-subscriber** and the **cloud-provider** returns authentication information that the **cloud-subscriber** can subsequently use to access the service.

Failure Conditions: (1) the **unidentified-user** does not provide a suitable name; (2) the financial information is not valid; (3) **cloud-provider** fails to notify the **cloud-subscriber** the account is open.

Failure Handling: For (1) and (2), new account is not created; For (3) See Use Case 3.2 below on failure handling related to notifications from **cloud-provider** to **cloud-subscriber**.

Requirements File: None.

Credit: TBD

3.2 Close An Account

Actors: **unidentified-user**, **cloud-subscriber, cloud-provider, payment-broker**.

Goals: Close an existing account for a **cloud-subscriber**.

Assumptions: A **cloud-subscriber** accesses a **cloud-provider**'s account management web page to receive information about closing an account. Account closure requires the date and time that the account should be closed as well as the method for disposition of data (returning data to **cloud-subscriber** and deletion of data from **cloud-provider**'s system). We assume one account per **cloud-subscriber**.

Success Scenario (close, IaaS, PaaS, SaaS): The **cloud-subscriber** interacts with the **cloud-provider**'s account management web page and requests that their account be closed on a particular date and time. The **cloud-subscriber** optionally requests the return of data stored with the **cloud-provider** to the **cloud-subscriber** (See Use Case "Copy Data Objects Out of a Cloud") or a transfer of data to a different **cloud-provider** (See Use Case "Copy Data Objects between Cloud Providers). In addition, the **cloud-subscriber** optionally specifies the data erase procedure to be performed by the **cloud-provider** after any copy operations have been performed (See Use case "Erase Data Objects in a Cloud"). The **cloud-provider**:(1) performs the requested actions on the timetable requested; (2) charges the **cloud-subscriber** according to the terms of the service; (3) notifies the **cloud-subscriber** that the account has been closed within an agreed to amount of time after the account closes; (4) deletes the **cloud-subscriber**'s **payment-broker** information from the **cloud-provider**'s systems; and (5) revokes the **cloud-subscriber**'s authentication information. Now the **cloud-subscriber** is classified as an **unidentified-user**.

Failure Conditions: (1) the **cloud-provider** closes the account too early or after the requested time and date; (2) an unauthorized user accesses a **cloud-provider**'s account management web page and impersonates the real **cloud-subscriber** and closes the account; (3) the requested disposition of data is not faithfully executed; (4) the **cloud-provider** does not completely delete the **cloud-subscriber**'s **payment-broker** information; (5) the **cloud-provider** overcharges the **cloud-subscriber**; (6) **cloud-provider** fails to notify the **cloud-subscriber** the account is closed; (7) **cloud-provider** fails to revoke the **cloud-subscriber**'s authentication information.

Failure Handling: For (1) the **cloud-subscriber** will need to contact the **cloud-provider** to reinstate the account if it was closed too early and, if too late, the **cloud-provider** may inadvertently give away free service; For (2) the **cloud-subscriber** will need to contact the **cloud-provider** and the **cloud-provider** will need to reinstate the account; For (3) only the **cloud-provider** will know unless a data leak is discovered by the **cloud-subscriber**. If that happens, **cloud-subscriber** must confront the **cloud-provider**. (See Use Case "Erase Data Objects In a Cloud"); For (4), only the **cloud-provider** will know unless the **cloud-subscriber** continues to be billed. If that happens, **cloud-subscriber** must confront **cloud-provider**. (See Use Case "Erase Data Objects In a Cloud"); For (5), **cloud-subscriber** must confront **cloud-provider**; For (6) **cloud-subscriber** should contact **cloud-provider** if time window for notification is exceeded; For (7) **cloud-provider** retries its revocation procedure.

Note: We might want to consider non-repudiation for some important **cloud-provider** messages; e.g., when an account gets closed, perhaps the **cloud-provider** should send a time-stamped and signed message to the former **cloud-subscriber** that asserts the means that were used to ensure that the **cloud-subscriber**'s data were completely removed (e.g., merely-unlinked, zero-writing memory/disk, n-pass overwrite). An efficient market would price these various erasure methods very differently. While such messages would not enforce erasure methods and could easily be faked, they would be hard evidence about the **cloud-provider**'s intended behavior and could serve as a basis for third party audits.

Requirements File:

Credit: TBD

3.3 Terminate An Account

Actors: **unidentified-user**, **cloud-subscriber, cloud-provider**.

Goals: **Cloud-provider** terminates a **cloud-subscriber's** account.

Assumptions: A **cloud-provider** determines that a **cloud-subscriber**'s account should be terminated per the terms of the SLA. The issue of multiple accounts for a **cloud-subscriber** is not considered part of the scope of this use case, nor is the issue of retaining sufficient information to recognize an abusive **cloud-subscriber** trying to create a new account to continue the abuse.

Success Scenarios: **(terminate, IaaS, PaaS, SaaS):** Possible reasons for termination may be that the **cloud-subscriber** has violated acceptable usage guidelines (e.g., by storing illegal content, conducting cyber attacks, or misusing software licenses), or that the **cloud-subscriber** is no longer paying for service. The **cloud-provider** sends a notice to the **cloud-subscriber** explaining the termination event and any actions the **cloud-subscriber** may take to avoid it (e.g., paying overdue bills, deleting offending content) or to gracefully recover data. Optionally, the **cloud-provider** may freeze the **cloud-subscriber**'s account pending resolution of the issues prompting the termination.

If the **cloud-subscriber** can pay for disposition of data currently stored in the **cloud-provider**'s system, and performing the requested disposition actions is legal, the **cloud-provider** performs the requested actions, charges the **cloud-subscriber** according to the terms of the service, notifies the **cloud-subscriber** that the account has been terminated, deletes the **cloud-subscriber**'s payment information from the **cloud-provider**'s system, and revokes the **cloud-subscriber**'s identity credentials. At this point, the **cloud-subscriber** becomes an **unidentified-user**.

Failure Conditions and Handling: If a cloud-subscriber prevents the termination by correcting the reason for termination (e.g., paying a late bill), then that could be seen as a failure of the use case in the sense that the termination does not occur.

Credit: Various cloud-provider SLAs and customer agreements

3.4 Copy Data Objects Into A Cloud

Actors: **cloud-subscriber, cloud-provider, transport-agent**.

Goals: **Cloud-subscriber** initiates a copy of data objects from the **cloud-subscriber's** system to a **cloud-provider's** system. Optionally, protect transferred objects from disclosure.

Assumptions: Assumes the Use Case "Open an Account" for **cloud-subscriber** on **cloud-provider**'s system. The **cloud-subscriber** has modify access to a named data object container on the **cloud-provider**'s system.

Success Scenario 1 (cloud-subscriber-to-network copy, IaaS, PaaS, SaaS): The **cloud-subscriber** determines a local file for copying to the **cloud-provider**'s system. The **cloud-subscriber** issues a command to the **cloud-provider**'s system to copy the object to a container on the **cloud-provider**'s system. The command may perform both the object creation and the data transfer, or the data transfer may be performed with subsequent commands. The command specifies the location of the local file, the data encoding of the local file, and the name of the new object within the container. If the **cloud-subscriber** requests protection from disclosure, cryptography is used to protect the objects in transit. The command returns the success status of the operation from the **cloud-provider**'s system to the **cloud-subscriber**. The **cloud-provider** charges the **cloud-subscriber** for the transfer according to the terms of the SLA, and begins accruing storage charges.

Failure Conditions 1: (1) partial writes and concurrent accesses; (2) size limitations, i.e., the local file will not fit into the container; (3) network fails repeatedly during transfer; (4) security breaches resulting in stolen data are discovered by **cloud-provider**; (5) data loss during transfer; (6) data errors during transfer; (7) **cloud-provider**'s system fails to notify the **cloud-subscriber** the successful data object transfer to container.

Failure Handling 1: For (1), (3), (5), (6), **cloud-subscriber** retries request; For (4) cloud-provider sends a notice of unauthorized disclosure to the **cloud-subscriber**; For (2), **cloud-subscriber** contacts **cloud-provider** for larger container; For (7), See Use Case "Close Account" on failure handling related to notifications from **cloud-provider** to **cloud-subscriber**.

Additional Assumptions: Data in transit is protected by one of two methods: 1) the **cloud-subscriber** encrypts data prior to copying it onto the disk drive and also informs the **cloud-provider** of the decryption key via a secure connection and the **cloud-provider** then decrypts the data before copying it into a new object, 2) the **cloud-subscriber** encrypts the data prior to copying it onto the disk drive and then, later, performs the decryption using processing resources of the cloud. The **cloud-provider** will provide disk drives to **cloud-subscriber** or will accept **cloud-subscriber**-provided disk drives.

Success Scenario 2 (cloud-subscriber-to-transport-agent copy, IaaS, PaaS, SaaS): The **cloud-subscriber** prepares a local file for copying to the **cloud-provider**'s system. The **cloud-subscriber** accesses the **cloud-provider**'s documentation and determines the characteristics of disk drives that the **cloud-provider** accepts for data import. The **cloud-subscriber** uses a **cloud-provider**-compatible disk and connects the disk drive to the **cloud-subscriber**'s computer system and performs a local copy of the local file onto the disk drive, along with a manifest specifying the encoding of the file, the container in which the file should be placed at the **cloud-provider**, access control metadata about the file, and the file's intended name. The **cloud-subscriber** uses a **transport-agent** to deliver the disk drive to the **cloud-provider**. On receipt of the disk drive, the **cloud-provider** connects the disk drive to the **cloud-provider**'s system and performs a local copy of the data into the container specified by the **cloud-subscriber**, and either retains or returns the disk drive according how the drive was provisioned. If the drive is to be re-used by the **cloud-**

provider, the **cloud-provider** erases all cloud-subscriber data on the disk using a suitable mechanism (see Use Case: "Erase Data Objects In a Cloud"), sends an attestation to the **cloud-subscriber** that the erase operations have been performed, and charges the **cloud-subscriber** if they requested special erase operations.

Failure Conditions 2: (1) **cloud-subscriber** sends inappropriate disk that fails to satisfy the requirements of the **cloud-provider**; (2) data object is in format not supported by **cloud-provider**; (3) **transport-agent** loses disk

Failure Handling 2: For (1) **cloud-provider** returns disk to **cloud-subscriber**; For (2) **cloud-provider** returns disk to **cloud-subscriber** and sends message to **cloud-subscriber** requesting data is resent in proper file encoding format; (3) **transport-agent** notifies **cloud-subscriber** of loss.

Requirements File:

Credit: This scenario is inspired by the Amazon S3 system.

3.5 Copy Data Objects Out of a Cloud

Actors: **unidentified-user**, **cloud-subscriber, cloud-provider**, **transport-agent**.

Goals: **Cloud-subscriber** initiates a copy of data objects from a **cloud-provider's** system to a **cloud-subscriber's** system. Optionally, protect transferred objects from disclosure.

Assumptions: The **cloud-subscriber** has "read" access to the objects and "traverse" access to object containers.

Success Scenario 1 (network-to-cloud-subscriber copy, IaaS, PaaS, SaaS): A **cloud-subscriber** prepares a local directory to receive a new file obtained from the **cloud-provider**'s system. The **cloud-subscriber** issues a command to the **cloud-provider**'s system to retrieve an existing object. The object resides in a container that itself resides on the **cloud-provider's** system, and the **cloud-subscriber** has "read" (or equivalent) access to the file as well as "traverse" (or equivalent) access to the container (and any containing containers). The **cloud-provider** authenticates the **cloud-subscriber's** identity using credentials (e.g., by verifying a signature generated using a private key held by the **cloud-subscriber**) that have been previously established, e.g., at account setup. The command specifies the unique identifier of the object to be copied, the location on the **cloud-subscriber's** system that will receive the object (which is called a file on the user's system), and the data encoding of the object (e.g., ASCII, GIF, ZIP). Either as part of the command or via a separate command, the **cloud-provider** generates a checksum value that can be used later to check that object contents were not altered in transit. Optionally, the command specifies that the object's content should be protected from disclosure during transit. The command returns the success status of the operation after the object has been copied. The **cloud-provider** charges the **cloud-subscriber** that owns the object for the data transferred according to the terms of service. Optionally, the **cloud-provider** charges the **cloud-subscriber** that made the request (if different from the owning **cloud-subscriber**).

Failure Conditions 1: (1) the object is corrupted in transit, or only part of it is received; (2) the object is disclosed in transit even though disclosure protection was requested; (3) the object is made inaccessible (e.g., moved, or "read" access removed by the object's owner) before the copy operation can begin (race condition).

Failure Handling 1: For (1), the **cloud-subscriber** retries the operation; For (2), the **cloud-provider** sends the **cloud-subscriber** a notice of unauthorized disclosure; For (3), the **cloud-subscriber** could retry the operation if the object has moved, but must contact the object's owner if access has been revoked.

Success Scenario 2 (network-to-unidentified-user copy, IaaS, PaaS, SaaS): An **unidentified-user** prepares a local directory to receive a new file obtained from the c**loud-provider**'s system. The **unidentified-user** issues a command to the **cloud-provider**'s system to retrieve an existing object. The object resides in a container that itself resides on the **cloud-provider's** system, and the **unidentified-user** has "read" (or equivalent) access to the file as well as "traverse" (or equivalent) access to the container (and any containing containers). The **cloud-provider** determines that the command originated from an unauthenticated entity (i.e., an **unidentified-user**). The **unidentified-user** will have the access rights that the **cloud-provider** offers to all **unidentified-users**. The command specifies the unique identifier of the object to be copied, the location on the **unidentified-user's** system that will receive the object (which is called a file on the **unidentified-user's** system), and the data encoding of the object (e.g., ASCII, GIF, ZIP). Either as part of the command or via a separate command, the **cloud-provider** generates a checksum value that can be used later to check that object contents were not altered in transit. The command returns the success status of the operation after the object has been copied. The **cloud-provider** charges the **cloud-subscriber** that owns the object for the data transferred according to the terms of service.

Failure Conditions 2: (1) the object is corrupted in transit, or only part of it is received; (2) the object is made inaccessible (e.g., moved, or "read" access removed by the object's owner) before the copy operation can begin (race condition).

Failure Handling 2: For (1), the **unidentified-user** retries the operation; For (2), the **unidentified-user** could retry the operation if the object has moved, but must contact the object's owner if access has been revoked.

Success Scenario 3 (physical-to-cloud-subscriber, IaaS, PaaS, SaaS): A **cloud-subscriber** accesses the **cloud-provider's** documentation and determines the characteristics of disk drives that the **cloud-provider** accepts for data export. The **cloud-provider** may provide disk drives to **cloud-subscribers** or may accept **cloud-subscriber**-provided disk drives. The **cloud-subscriber** obtains a **cloud-provider**-compatible disk. The **cloud-subscriber** writes a manifest onto the disk drive that specifies the location of the objects in the cloud to be copied onto the disk drive, and whether the objects should be encrypted prior to shipping to protect their confidentiality. If the **cloud-provider** is providing the disk drive, this information may be sent over the network instead. If the **cloud-subscriber** is providing the disk drive, the **cloud-subscriber** uses a **transport-agent** to deliver the disk drive to the **cloud provider.** Once the **cloud-provider** has the disk drive either by receipt from the **transport-agent** or by procurement, the **cloud-provider** connects the disk to the cloud system, computes checksums on the data objects to be transferred, optionally encrypts data objects to be transferred, performs a local copy of the specified data objects onto the disk drive, and uses a **transport-agent** to send the disk drive to the **cloud-subscriber**. The **cloud-provider** conveys the checksums and key material for decrypting the contents using a different channel that is itself protected using the **cloud-subscriber's** credentials (e.g., a public key known to the **cloud-provider**). The **cloud-subscriber** takes steps to safeguard the key materials from loss (e.g., backup on stable storage). On receipt of the disk drive, the **cloud-subscriber** connects the disk drive to the **cloud-subscriber**'s computer system and performs a local copy of the data objects to the **cloud-subscriber**'s computer system. If encryption was requested, the **cloud-subscriber** decrypts the objects using the key material indicated by the **cloud-provider**. The **cloud-subscriber** validates checksums on the objects. Depending on the provisioning of the disk drive, the **cloud-subscriber** may return it to the **cloud-provider**.

Failure Conditions 3: (1) a **cloud-subscriber**-provided disk is lost before arriving at the **cloud-provider** or is defective; (2) the disk is lost or damaged in transit from the **cloud-provider** to the **cloud-subscriber**; (3) data objects on the disk received by the **cloud-subscriber** are corrupted; (4) the key material and/or checksum information is lost before it can be received by the **cloud-subscriber**.

Failure Handling 3: For (1) and (2), procure a new disk and retry. For (3) and (4), retry.

Requirements File: NA

Credit: The idea of charging the owning **cloud-subscriber** or the requesting **cloud-subscriber** is from Amazon. The idea of using a disk for bulk transfer is inspired by Amazon.

3.6 Erase Data Objects In a Cloud

Actors: **unidentified-user**, **cloud-subscriber, cloud-provider**.

Goals: Erase a data object on behalf of a **cloud-subscriber** or **unidentified-user**.

Assumptions: One or more data objects already exist in a **cloud-provider**'s system. A request to erase a data object includes the unique identifiers of the objects to delete, date and time when the deletion should occur, and the means that the **cloud-provider** should employ to perform the deletion operation (e.g., simply returning the space for use by others, zero-filling the object prior to return, n-pass overwriting of the object with random data). There is no redundant data storage by **cloud-provider** or redundant copies are deleted together.

Success Scenario 1 (erase, IaaS, PaaS, SaaS): A **cloud-subscriber** (or **unidentified-user** if they have been granted access to a container/object) sends a delete-objects request to the **cloud-provider**'s system. At the requested deletion time, the **cloud-provider** disables all new attempts to access the object. The **cloud-provider** continues to perform in-process data transfers for the object. When all current data transfers have completed or timed out, the **cloud-provider** performs the requested deletion operation on the media that stored the object, charges the **cloud-subscriber** for the service, and then sends back to the **cloud-subscriber** a time-stamped, signed message attesting to the steps that have been taken to delete the object within an agreed to period of time after deletion.

Failure Conditions: (1) the object is moved or renamed before the deletion operation is attempted (race condition); (2) **cloud-provider** erases an incorrect data object; (3) an unauthorized user accesses a **cloud-provider**'s account management web page and impersonates the real **cloud-subscriber** and requests the data deletion which then occurs; (4) access to the object is disabled before date and time requested by **cloud-subscriber**; (5) **cloud-provider** fails to notify the **cloud-subscriber** that the object is erased; (6) erasure of the object is not performed completely or at all by **cloud-provider**.

Failure Handling: For (1) the **cloud-provider** should receive an error message from the attempted erasure and should retry; For (2) the **cloud-subscriber** should notify the **cloud-provider** and the **cloud-provider** should undo deletion on wrong data and perform deletion on the correct data object; For (3) the **cloud-subscriber** should notify the **cloud-provider** and the **cloud-provider** should undo the deletion; For (4) the **cloud-subscriber** must contact the **cloud-provider** to undo erasure; For (5) the **cloud-subscriber** must query the **cloud-provider** to ask if the deletion did occur – if not, the **cloud-provider** must retry the delete operation immediately; For (6) the **cloud-subscriber** must contact the **cloud-provider** and the **cloud-provider** must delete immediately or reattempt deletion.

Requirements File:

Credit: TBD

3.7 VM Control: Allocate VM Instance

Actors: **cloud-subscriber, cloud-provider**

Goals: The **cloud-subscriber** should have the capability to create VM images that meet its functions, performance and security requirements and launch them as VM instances to meets its IT support needs.

Assumption: The **cloud-subscriber** has an account with an IaaS cloud service that enables creation of Virtual Machine (VM) images and launching of new VM instances. The **cloud-provider** shall offer the following capabilities for VM Image creation to the **cloud-subscriber**:

1) A set of pre-defined VM images that meets a range of requirements (O/S version, CPU cores, memory, and security)

2) Tools to modify an existing VM image to meet **cloud-subscriber**'s requirements

3) Tools to create a new VM image from scratch

The **cloud-provider** shall support the following capabilities with respect to launching of a VM instance:

1) Secure launching of a VM instance (e.g., enabling creation of an asymmetric cryptographic key pair)

2) Secure administration of the **cloud-subscriber**'s VM instance through the ability to:

4. configure certain ports (e.g., opening of port 22 for enabling a SSH session;
5. allow **cloud-subscriber's** scanning tools on the launched VMs for presence of appropriate patches (based on Guest O/S) or absence of malware

3) Cloud-subscriber shall be able to suspend and re-start VM instances

Success Scenario: (<u>AllocateVM</u>, IaaS): (1) The **cloud-subscriber** requests a specific pre-defined Virtual Machine image supplied by the **cloud-provider** (O/S, CPU cores, memory, and security) and launches new VM instances. (2) The **cloud-subscriber** is able to modify a VM image according to their requirements using **cloud-provider's** tools. (3) The **cloud-subscriber** has secure launching and administration of their VM instance.

Failure Condition: (1) The **cloud-subscriber** is not able to successfully complete a request to create a Virtual Machine from **cloud-provider's** inventory; (2) The **cloud-subscriber** is not able to modify or create a Virtual Machine image according to their specifications with the **cloud-provider's** toolset; (3) The **cloud-subscriber** is not able to invoke their required security protections on their VM image/VM instance.

Failure Handling: (1) The **cloud-provider** must verify that the request made by the **cloud-subscriber** is valid and then take corrective steps to assist the **cloud-subscriber** or take necessary action to provide the VM configuration; (2) The **cloud-provider** must verify correct usage of their toolset, assist the **cloud-subscriber** or allow the **cloud-subscriber** to use their own methodology for VM creation; (3) On receipt of a security error message, the **cloud-subscriber** retries the operations; on multiple failures, the **cloud-subscriber** contacts the **cloud-provider** for resolution of the failure.

Credit: Original use case derived from features of Amazon Web Services; we also note the applicability of OpenNebula, OpenStack, CloudStack and other cloud framework products.

3.8 VM Control: Manage Virtual Machine Instance State

Actors: **cloud-subscriber**, **cloud-provider**

Goals: A **cloud-subscriber** stops, terminates, reboots, starts or otherwise manages the state of a virtual instance

Assumptions: A suitable VM image (operating system executables and configuration data) exists. Possible formats include OVF.

Success Scenario 1 (start-stop-non-persistent-VMs, IaaS): A **cloud-subscriber** identifies a VM image to run. The **cloud-subscriber** chooses a number of VMs and issues a command to load the VM image into the chosen number of VMs and execute. The **cloud-provider** provisions VMs and performs the loading and boot-up cycle for the selected image for the requesting **cloud-subscriber** and initializes each VM with the cloud-**subscriber**'s credentials (so the **cloud-subscriber** can log in). The provisioning includes the allocation of an IP address. The boot device (root file system) for each VM is non-persistent. The **cloud-subscriber** may issue commands that connect persistent media as non-root file systems or non-file system devices for each of the VMs, operate the VMs to read or store data onto those devices, and then stop the VMs. Upon a VM's exit, the contents of the boot device are lost but data written to other devices during the run is preserved. The IP address for the VM is disassociated when the VM is stopped. The **cloud-provider** charges the **cloud-subscriber** for cpu time, storage time, network usage, and possibly for system startup cycles.

Failure Conditions: (1) The VM image may fail to boot correctly; (2) VMs may fail to stop on command. (Note that many network-level failures could be enumerated like, *e.g*., fails-to-obtain-a-valid-IP-address.)

Failure Handling: For (1), the **cloud-subscriber** can choose a different VM image, or debug; for (2) the **cloud-subscriber** can request the **cloud-provider** to terminate the stalled VMs.

Success Scenario 2 (start-stop-persistent-VMs, IaaS): A **cloud-subscriber** identifies a VM image to run. The **cloud-subscriber** chooses a number *N* of VMs and issues a command to load the image onto a persistent media (most likely a form of network-attached storage). The **cloud-subscriber** issues a command to boot *N* VMs from the persistent media, using the **cloud-subscriber's** credentials for each (so the **cloud-subscriber** can log in). The **cloud-provider** provisions *N* VMs, associates the persistent network storage with each as a boot device and initiates the boot sequence. The boot device is persistent and the data contents survive VM shutdowns. The **cloud-subscriber** may issue commands that connect additional persistent media as non-root file systems or non-file system devices for each VM, operate the VMs to read or store data onto those devices, and then stop the VMs. Upon a VM's exit, the contents of all persistent devices are preserved. The IP address for the VM is disassociated when the VM is stopped. The VMs can be restarted on command. The **cloud-provider** charges the **cloud-subscriber** for cpu time, storage time, network usage, and possibly for system startup cycles. The VMs can be restarted.

Failure Conditions 2: (1) The VM image may fail to boot correctly; (2) the intended persistent boot device may fail; (3) VMs may fail to stop on command. (Note that many network-level failures could be enumerated like, *e.g*., fails-to-obtain-a-valid-IP-address.)

Failure Handling 2: For (1), the **cloud-subscriber** can choose a different VM image, or debug; for (2) the **cloud-subscriber** can retry or consult with the **cloud-provider**; for (3) the **cloud-subscriber** can request the **cloud-provider** to terminate the stalled VMs.

Requirements File: NA

Credit: Original use case derived from features of Amazon Web Services; we also note the applicability of OpenNebula, OpenStack, CloudStack and other cloud framework products.

3.9 Query Cloud-Provider Capabilities and Capacities

Actors: **cloud-user, cloud-provider**

Goals: A **cloud-user** makes a structured capability or capacity or price request to one or several **cloud-providers** and receives a structured response that can be used as input to drive service decisions.

Assumptions: An extensible industry request/response interface for **cloud-provider** capability and capacity characteristics. The capability request format will include a minimum set of named capabilities that all implementers of the standard can respond to affirmatively or negatively. Each named capability has related specific metric parameters, for example size, speed and number of cores for an instance specification, compliance or noncompliance for a Trusted Internet Connection (TIC) specification, or block or REST interface for a storage specification. Capability responses might include **cloud-provider**-specific identifiers for the returned capability to assist in subsequent capacity queries or provisioning requests. Queries on capabilities beyond the minimum agreed set are permitted, and return an “unrecognized-capabilities” response. The capacity request format and interface is dependent upon capability definition, with parameters such as capability of interest, and metric desired. Examples include availability of a quantity of instance capabilities of a given size, guaranteed response time, and available storage volume for a specified storage type. Capacity responses contain a pricing response with a time window of availability for the pricing so that **cloud-user**s can make procurement decisions. Capacity responses do not guarantee that actual availability will continue beyond the moment of query.

Success Scenario 1 (Capability Request: IaaS, PaaS, SaaS): A **cloud-user** wishes to determine whether **cloud-provider-1** can support a named cloud capability, for example the ability to run instances of a specified size and speed, the ability to support queuing, or the ability to supply a particular named class of storage. The **cloud-user** marshals a capability request using an industry-standard format, and transmits the request to **cloud-provider-1** using an industry-standard request/response method and **cloud-provider-1**’s authentication credentials. **Cloud-provider-1** receives the capabilities request, evaluates it against its capabilities data and returns a structured response to the **cloud-user** specifying the extent to which the desired capability is available from **cloud-provider-1**. Responses can be affirmative, negative, or “near miss” responses containing structured variance data to assist in fallback decisions. The **cloud-user** evaluates the response, either through an automated program or human review, and makes an allocation decision or subsequent capacity requests for **cloud-provider-1**. A capability request can be repeated across multiple **cloud-provider**s to compare capabilities at a point in time to drive service acquisition or allocation decisions.

Failure Conditions 1: (1) Network, authentication or interface difficulties cause the request to fail; (2) **Cloud-provider-1** request processing fails; (3) **Cloud-provider-1** is unable to recognize the named cloud capability because it is outside the minimum required capabilities.

Failure Handling 1: (1) The **cloud-user** observes request failure and consequent error messages and retries in the case of a transient error or remedies problem; (2) The **cloud-user** observes request failure and retries in the case of a transient error or contacts **cloud-provider-1**; (3) **Cloud-provider-1** returns a specific “unrecognized capability” response indicating that the requested capability is not a recognized capability for **cloud-provider-1**.

Success Scenario 2 (Capacity Request: IaaS, PaaS, SaaS): A **cloud-user** wishes to ascertain whether **cloud-provider**-1 has adequate capacity to provide a named capability and the current cost of the capacity/capability pair. The **cloud-user** marshals a capacity request using an industry-standard format and transmits the request to **cloud-provider-1** using an industry-standard request/response method and **cloud-provider-1** authentication credentials. **Cloud-provider-1** receives the capacity request, evaluates it against their current runtime availability data and spot pricing information, and returns a structured response to the **cloud-user** specifying the extent to which the desired capacity is available from **cloud-provider-1**, the current cost at which it is available to the requesting **cloud-user**, and the time window within which the capacity is available at the specified cost. Note that the time window guarantees pricing only; actual availability is not guaranteed beyond the moment of request. The **cloud-user** evaluates the response, either through an automated program or human review, and makes an allocation decision for **cloud-provider-1**. A capacity request can be repeated across multiple **cloud-**

providers to compare capabilities at a point in time to drive service acquisition or allocation decisions. Often a capacity request will follow a previous capability request/response invocation and contain the **cloud-provider-1** specific identifier for the affirmatively-queried capability as a parameter.

Failure Conditions 2: Include all **Scenario 1:** capabilities failure conditions with respect to capacities.

Failure Handling 2: Include all **Scenario 1:** capabilities failure handling with respect to capacities.

Success Scenario 3 (Set Request, IaaS, PaaS, SaaS): A **cloud-user** wishes to determine if all or part of a set of capabilities or capacities are available from **cloud-provider-1**. For example, the **cloud-user** might request capacity for a set consisting of 20 VM's of specified capability, 500GB of block storage, 100 GB/month network bandwidth and TIC compliance. The **cloud-user** marshals and sends **cloud-provider-1** a capacity or capability request as discussed in previous scenarios, using an extended syntax that permits assembly of multiple capabilities or capacities into a single request. **Cloud-provider-1** evaluates the capacity or capability request's components and returns a structured response for the set which contains an overall capability or capacity response for the set containing per cent coverage of components, along with information on each component in the set. The **cloud-user** receives the response and evaluates set coverage information to make an allocation or procurement decision.

Failure Conditions 3: (1) Include all **Scenario 1:** capabilities failure conditions with respect to capacities; (2) Elements of capability or capacity set are not recognized by cloud-provider-1 because they fall outside the minimum required capabilities; (3) Processing failure on **cloud-provider**-1 fails to return a valid result for one of capability or capacity set.

Failure Handling 3: (1) Include all **Scenario 1:** capabilities failure handling with respect to capacities; (2) Elements of capability or capacity set falling outside minimum required capabilities deliver an "unrecognized capability" response within the set, and are treated as "not-implemented" by the overall response; (3) Elements of capability or capacity set failing to process deliver an error response within the set and are treated as "not-implemented" by the overall response.

Requirements File: NA

Credit: NA

4. Cloud Interoperability Use Cases

The following use cases have been prepared to address scenarios related to cloud interoperability. At the present time, they represent a substantial sampling of typical cloud ineroperability use cases, focusing specifically on transfer of data and virtual machine contents between clouds, but do not yet comprise a complete or comprehensive set. Work is ongoing within the SAJACC group to identify a structure that can provide a more comprehensive set of interoperability features based on community input, existing standards and cloud product application programmer interface (API) capabilities, the NIST Cloud Computing Reference Architecture and existing best practices for cloud interoperability. It should be noted here that cloud brokerage and cloud federation are specific areas in which standards and the capabilities of cloud products that are specifically oriented to these areas will have strong applicability.

4.1 Copy Data Objects between Cloud-Providers

Actors: **cloud-subscriber, cloud-provider-1, cloud-provider-2**, **transport-agent**

Goals: Copy data objects from a **cloud-provider**-1's system to a **cloud-provider-2**'s system on the initiative of a **cloud-subscriber**.

Assumptions: Cloud-subscriber has established an account with **cloud-provider-1 and cloud-provider-2.**

Success Scenario (copy, IaaS): A **cloud-subscriber** mutually authenticates to **cloud-provider-1** (where the data object initially resides) using **cloud-provider-1**'s mutual authentication mechanisms, and starts a command shell (or equivalent) on **cloud-provider-1**. From **cloud-provider-1**, the **cloud-subscriber** may access other systems on the Internet. The **cloud-subscriber** determines the object identifiers of the data objects that the **cloud-subscriber** wishes to copy from **cloud-provider-1 to cloud-provider-2**. From the command shell on **cloud-provider-1** the **cloud-subscriber** authenticates to **cloud-provider-2** using **cloud-provider-2**'s authentication mechanisms (note: this approach passes authentication through **cloud-provider-1**). The **cloud-subscriber** locates a container (e.g., a directory) on **cloud-provider-2** where the copied object will reside. The **cloud-subscriber** may have to create a container. For each data object that the **cloud-subscriber** wishes to copy, the **cloud-subscriber**: 1) downloads the contents of the object to the virtual machine the **cloud-subscriber** is using in **cloud-provider-1** 2) uploads the data as a new object in **cloud-provider-2**'s object store, and 3) deletes the copy of the data just created in the virtual machine in **cloud-provider-1**. The copy of the data just created in virtual machine in cloud-provider-1 is deleted as described in Use Case 3.6 (Erase Data Objects in Clouds).

Failure Conditions: (1) The **cloud-subscriber** is unable to authenticate to **cloud provider-1**; (2) the **cloud-subscriber** has insufficient privileges for the requested actions**.**

Failure Handling: The **cloud-providers** notify the subscriber of the failure and provide a description of the failure (e.g. expired certificate, insufficient privileges, etc.).

Credit: TBD

Note: Success Scenario 3 or New Use Case – Version Control: further work is needed to explore topics related to the idea of several versions of same data object copied across multiple clouds and version control, and the related importance of distributed revision control systems.

4.2 Dynamic Dispatch to an IaaS Cloud

Actors: **cloud-subscriber, cloud-provider-1**, **cloud-provider-2,** … **cloud-provider-n**

Goals: Invoke operations on the most effective clouds available based on a client-side set of rules that are evaluated at runtime.

Assumptions: The **cloud-subscriber** has already established accounts with multiple IaaS **cloud-providers**.

Success Scenario (dispatch, IaaS): This use case is for workloads that do not depend on unique resources of a specific **cloud-provider**. The **cloud-subscriber** wishes to perform a job on the cloud that can offer the best performance, with the greatest reliability, at the least cost. From the time when the **cloud-subscriber** opened the account with each **cloud-provider**, the **cloud-subscriber** has a record of each **cloud-provider**'s service charges and promised performance and availability. Optionally, the **cloud-subscriber** queries each **cloud-provider** for any updates to the SLA regarding these issues and, if there are changes, evaluates the acceptability of the changes as in "Compare Service Level Agreements to Respond to a Change". Then the **cloud-subscriber** formulates a small test workload, which could have processing aspects, data storage aspects, or network performance aspects. The **cloud-subscriber** runs the test workload one or more times on each **cloud-provider**, and sorts the **cloud-providers** by availability, correctness of the workload's outputs, and performance. Alternatively, the **cloud-subscriber** queries the **cloud-providers** for performance, usage, availability, and cost metrics, and dispatches workloads accordingly. In this case, the **cloud-provider** bears the responsibility to maintain the needed querying interface.

Failure Conditions: (1) The **cloud-provider** is unable to provide the quality-of-service required for the dispatched workload; (2) the **cloud-provider** cannot scale to meet the **cloud subscriber**'s demand; (3) **cloud-provider** is unable to provide meaningful metrics. .

Failure Handling: **Cloud-subscriber** dispatches workload to another **cloud-provider**.

Credit: This use case was inspired by the libcloud project [LIBCLOUD], which provides a client-side library for interacting with multiple IaaS **cloud-providers** concurrently using a single API. Other related concepts have been presented by the CompatibleOne open-source cloud brokerage project.

4.3 Cloud Burst From Data Center to Cloud

Actors: **cloud-subscriber, cloud-provider, cloud-management-broker**

Goals: Maintain required service levels for an agency's data-center hosted process, by dynamically allocating/de-allocating cloud computer or storage resources to service current demands.

Assumptions: Assumes the Use case "Open an Account"

Success Scenario 1 (base, IaaS): **Cloud-subscriber** provisions and maintains **cloud-provider-1** virtual machine images and/or configured storage capacity designed to support **cloud-subscriber**-defined units of work ranging in scope from individual computing or storage tasks to entire distributed applications. **Cloud-subscriber** establishes load monitoring processes for the units of work concerned, and load threshold and sensitivity limits for cloud bursting. Upper limits govern starting new processes on **cloud-provider** to handle increasing load; lower limits govern stopping **cloud-provider** processes to handle decreasing load. As monitored load triggers threshold limits, processes start or stop on **cloud-provider-1** infrastructure to maintain required service levels.

Failure Conditions 1 (base): Failed allocation or de-allocation event

Failure Handling 1 (base): Failed allocation or de-allocation event: **Cloud-provider-1** notifies **cloud-subscriber**. **Cloud-subscriber** either communicates with **cloud-provider-1** for resolution within an acceptable SLA or has access to automated **cloud-provider-1** notification and resolution. Failed de-allocation events can result in excess agency charges and must be covered in SLA agreements.

Success Scenario 2 (Manual Bursting, IaaS): **Cloud-subscriber** manually allocates and de-allocates **cloud-provider** resources based on threshold notifications.

Failure Conditions 2 (Manual Bursting): N/A

Failure Handling 2 (Manual Bursting): N/A

Success Scenario 3 (Automated Bursting, IaaS): **Cloud-management-broker** processes monitor load and threshold limits and allocate or de-allocate **cloud-provider-1** resources using programming interfaces provided by **cloud-provider**.

Failure Conditions 3 (Automated Bursting): Failed event detection

Failure Handling 3 (Automated Bursting): Cloud-management-broker independently monitors its event detection services and notifies **cloud-subscriber** of outages so **cloud-subscriber** can fall back to manual bursting scenarios.

Credit: N/A

4.4 Migrate a Queuing-Based Application

Actors: **cloud-subscriber, cloud-provider-1, cloud-provider-2, cloud-management-broker**

Goals: Migrate an existing queue and associated messages from one **cloud-provider** to another

Assumptions: cloud-subscriber is responsible for modifying applications accessing queues to access new queue after migration.

Success Scenario (IaaS): A **cloud-subscriber** wishes to migrate a **cloud-provider**-1 queue and its associated current messages to **cloud-provider**-2. Both **cloud-provider**-1 and **cloud-provider**-2 implement an agreed minimum set of message attributes, queue attributes and queue operations to facilitate migration activities. **Cloud-subscriber** issues a command to **cloud-management-broker** to migrate queue X on **cloud-provider**-1 to queue Y on **cloud-provider**-2. **Cloud-management-broker** issues commands using native API to **cloud-provider**-2 to create queue Y. **Cloud-management-broker** issues commands using native API to **cloud-provider**-1 to stop queue X processing in order to create a steady state. **Cloud-management-broker** issues commands to **cloud-provider**-1 to access messages in queue X and commands to **cloud-provider**-2 to create identical objects on queue Y using agreed minimum attribute set. **Cloud-provider** issues a start command to Queue Y and notifies **cloud-subscriber**.

Failure Conditions: (1) **Cloud-provider** is unable queuing operations; (2) **cloud-provider** cannot provide sufficient information in a timely manner about the status of queues.

Failure Handling: The **cloud-provider** notifies the **cloud-subscriber** of the failure and provides a description of the failure.

Credit: This use case inspired by Amazon's simple queuing service: http://aws.amazon.com/sqs. Several other cloud products contain similar concepts.

4.5 Migrate (fully-stopped) VMs from one cloud-provider to another

Actors: **cloud-subscriber**, **cloud-provider**-1, **cloud-provider**-2, **cloud-management-broker**

Goals: Seamlessly migrate an arbitrarily designated stopped virtual machine from **cloud-provider-1** to **cloud-provider-2**.

Assumptions: Include**s** the Use cases "Open An Account", "VM Control: Manage Virtual Machine Instance State", "VM Control: Allocate VM Instance". Both **cloud-provider**s are using para-virtualized devices, or are using identical hardware.

Success Scenario (migrate instance, IaaS): The **cloud-subscriber** issues commands to halt the source VM instance on **cloud-provider-1.**

Cloud-provider-1 generates a configuration file for the halted VM. The configuration file includes an abstract description of the virtual hardware provided by **cloud-provider-1** as well as a description of the storage devices needed for the VM to operate. Candidate fields for the configuration file include:

- The number of virtual CPUs
- The amount of memory used/assumed by the VM
- The unique hostname and IP address
- The Domain Name System resolver configuration used by the VM
- The list of virtual network interfaces used/assumed by the VM
- The subnet mask and identifier for each subnet attached to the VM
- The MAC address assigned to the VM
- The list of virtual block devices the VM assumes
- The list of attached storage devices (local or network-accessed file systems)
- The configuration file may take advantage of the OVF format or the Mirage Image Format, or others.

The **cloud-subscriber** submits the configuration file to **cloud-provider-2** and requests a translation into **cloud-provider-2's** environment. **Cloud-provider-2** returns a list of the fields in the configuration file that have reliable translations in **cloud-provider-2's** environment. The **cloud-subscriber** identifies any missing translations in the configuration file. If the **cloud-subscriber** cannot supply missing translations, the migration is cancelled.

Otherwise, the **cloud-subscriber** substitutes **cloud-provider-2's** translations in the configuration file, and, if the VM's root storage device is persistent, copies the data object representing the VM's root storage device to **cloud-provider-2** (See Use Case "Copy Data Objects Between Clouds"). If non-root storage devices are network accessible from outside of **cloud-provider-1's** infrastructure, the **cloud-subscriber** may choose to leave the data at **cloud-provider-1** and access them remotely; however access latencies may limit the availability of this approach. If non-root storage devices are not network accessible, or if the **cloud-subscriber** determines that the performance of remote access storage devices would not be sufficient, the **cloud-subscriber** copies the data objects containing the attached devices from **cloud-provider-1** to **cloud-provider-2**. (Note that this can be a large operation and the **cloud-subscriber** may choose to reconfigure the VM to avoid some of the copying.)

The **cloud-subscriber** issues VM management commands for **cloud-subscriber-2** to initialize a new VM in **cloud-subscriber-2** that is based on the information transferred from **cloud-provider-1** (See Use case "VM Control: Allocate VM Instances"). The new VM can now be managed at **cloud-provider-1** (See Use case "VM Control: Manage Virtual Machine Instance State").

Failure Conditions (migrate instance): Necessary translations in the VM's configuration file are missing for **cloud-provider-2.**

Failure Handling (migrate instance): There is no recovery except to choose a different **cloud-provider** as **cloud-provider-2**.

Credit: Success Scenario 2: Amazon AMI images. DMTF, "Open Virtualization Format Specification"; DMTF, DSP-IS0103_1.0.0: "Use Cases and Interactions for Managing Clouds", June 2010; "Opening Black Boxes: Using Semantic Information to Combat Virtual Machine Image Sprawl", D. Reimer et al, VEE '08 March 5-7, 2008, Seattle, Washington.

5. Cloud Security Use Cases

The area of cloud security is of particularly intense interest and importance. As a result, a large amount of associated work has been carried out by the NIST Cloud Computing Security Working Group, most of which has been completed after the preparation of the following SAJACC use case scenarios. The use cases presented below were designed to initiate discussion on making the connections that are needed between general security design considerations and testable use case scenarios that are formulated in the same way as other work presented here.

A later section of this report makes connections between the output of the NIST Security and other working groups and the continuation of the SAJACC validation process.

5.1 Identity Management - User Account Provisioning

Actors: **cloud-subscriber**, **cloud-subscriber-administrator**, **cloud-provider**

Goals: The **cloud-subscriber** requires to provision (create) user accounts for **cloud-subscriber-users** to access the cloud. Optimally, the **cloud-subscriber** requires the synchronization of enterprise system-wide user accounts from enterprise data center-based infrastructure to the cloud, as part of the necessary process to streamline and enforce *identical* enterprise security (i.e., authentication and access control policies) on **cloud-subscriber-user**s accessing the cloud.

Assumption: The **cloud-subscriber** has well defined policies and capabilities for identity and access management for its enterprise IT applications and data objects. The **cloud-subscriber** has enterprise infrastructure to support the export of **cloud-subscriber-user** account identity and credential data. The **cloud-provider** has identity provider (IdP) capabilities and has provided an interface (Web browser-based user interface or an API set) to accept the **cloud-subscriber**'s input and/or upload of **cloud-subscriber-user** identity data for account provisioning. The **cloud-subscriber** can establish trusted connections to these cloud services.

Success Scenario 1 (IaaS): This scenario illustrates how a **cloud-subscriber** can provision user/administrator accounts (mainly IT administrators, e.g., billing manager, system administrator, network engineer, etc.) on the IaaS cloud.

Steps: The **cloud-subscriber-administrator** gathers user identity and credential information (could be an extract or export from the enterprise's identity management store) and the account provisioning policies, including user privilege settings, such as user group/role assignment information. Optionally, the **cloud-subscriber-administrator** transforms and formats the provisioning data into the format required by the **cloud-provider**. The **cloud-subscriber-administrator** uses an identity management tool provided by the **cloud-provider**, through a Web browser-based user interface, a command line tool, or a set of identity management APIs, to input/upload the account provisioning data for the **cloud-subscriber**. Optionally, the **cloud-subscriber-administrator** uses the **cloud-provider**'s interface (Web browser-based, command line, or APIs) to configure access control policies of the new user accounts provisioned, ensuring enterprise dictated access policies are in place in the cloud and can be leveraged by the authentication and access control mechanism deployed in the cloud.

Success Scenario 2 (PaaS, SaaS): This scenario illustrates how a **cloud-subscriber** can provision end user accounts in the cloud, often in bulk fashion. The user identity and credential data are often readily available from the enterprise's identity management store.

Steps: The **cloud-subscriber-administrator** gathers user identity and credential information (often an extract or export from the enterprise's identity management store) and the security policies data, including user privilege settings, such as user group/role assignment information. Optionally, the **cloud-subscriber-administrator** transforms and formats the identity data into a standard-compliant format, such as SPML. The **cloud-subscriber-administrator** uses an identity management tool provided by the cloud-provider, through a Web browser-based user interface, a command line tool, or a set of identity management APIs, to upload the bulk account provisioning data for the **cloud-subscriber-users**. The **cloud-provider**'s identity management capabilities are now configured with the **cloud-subscriber-user** account data and the **cloud-subscriber**'s access control policy is now in place to be enforced.

Failure Condition/Failure Handling: TBD (User identity meta-data information from the enterprise doesn't meet **cloud-provider**'s requirements, etc.)

Credit: Cloud Security Alliance's Guidance for Identity and Access Management, V2.1; Amazon AWS Identity and Access Management (IAM) tools and documentation.

5.2 Identity Management - User Authentication in the Cloud

Actors: **cloud-subscriber**, **cloud-subscriber-user**, **cloud-provider**, **identity-provider** (optional)

Goals: The **cloud-subscriber-users** should be able to authenticate themselves using a standard-based protocol, such as SAML, OpenID or Kerberos, to gain access to the cloud application/service. Alternatively, the **cloud-subscriber-user** should be able to transparently log in to the cloud application/service once they are authenticated against any system that's part of single-sign-on federation of systems.

Assumption: The **cloud-subscriber-user**'s account has been already provisioned in the cloud, see use case **Identity Management – User Account Provisioning**. In the case of single-sign-on, prior trust relationships have been established (e.g., using trusted crypto keys) among the identity provider/authentication service and the cloud applications/services that are sharing the federated identity attributes of authenticated users.

Success Scenario 1 (PaaS, SaaS): This scenario illustrates how a **cloud-subscriber-user** can authenticate against a cloud-based authentication service using the appropriate credentials to gain access to the cloud-based applications/services.

Steps: The **cloud-subscriber-user** provides his/her credentials (e.g., using password tokens or smart card) to the **cloud-provider's** authentication service interface. The authentication request gets validated by the authentication service and an appropriate authentication token is issued using a standard-based protocol (such as a SAML authentication assertion). The **cloud-subscriber-user** then accesses cloud-deployed applications/services using the authentication token until the authenticated session expires or the user explicitly logs out using the authentication service' logout interface.

Success Scenario 2 (PaaS, SaaS, Single-Sign-On): This scenario illustrates how a **cloud-subscriber-user** authenticates against an authentication service (identity provider deployed either in the cloud or within the enterprise's IT infrastructure) and transparently gains access to cloud applications/services without presenting authentication credentials again, achieving single-sign-on.

Steps: The **cloud-subscriber-user** authenticates against the enterprise's authentication service/identity provider, obtains an authentication token (such as a digitally signed SAML authentication assertion); the **cloud-subscriber-user** accesses (through Web browser) applications/services deployed in the cloud with the authentication token; the authentication sub system provided by the **cloud-provider** transparently trusts the authentication token and obtains the federated identity attributes for access control decisions.

Failure Condition/Failure Handling: trust relationship among cloud-provider's services and the identity provider is not established;

Credit: Cloud Security Alliance's Guidance for Identity and Access Management, V2.1

5.3 Identity Management - Data Access Authorization Policy Management in the Cloud

Actors: **cloud-subscriber**, **cloud-subscriber-user**, **cloud-subscriber-administrator**, **cloud-provider**, **identity-provider** (optional)

Goals: A **cloud-subscriber-administrator** should be able to manage (*add/delete/change*) data access authorization policies for data stored in the cloud. Note: this capability is essential to fulfill the use case of **Sharing of access to data in a cloud.**

Assumption: The **cloud-subscriber-user** account has been already provisioned in the cloud, see use case **Identity Management – User Account Provisioning**. The **cloud-provider** has data access authorization mechanisms in place to use the authorization policies managed by the **cloud-subscriber-administrator**.

Success Scenario (IaaS, PaaS): **Steps:** The **cloud-subscriber-administrator** authenticates and logs on to the **cloud-provider**'s data access authorization policy tool (such as a command line tool to manage access to file system data objects in the cloud, or a Web interface to manage authorization policies to access data in a database). The **cloud-subscriber-administrator** executes commands or performs actions to create/change data access policies, e.g., change the ACL of a file system object. Optionally, the **cloud-subscriber-administrator** uploads prepared access authorization policies (such as encoded in XACML format) to the **cloud-provider**'s bulk policy management interface. Immediately following the update, the affected **cloud-subscriber-user** will be able to access a data object or be denied access to a data object depending upon the new policy.

Failure Condition/Failure Handling:

Credit:

5.4 Identity Management - User Credential Synchronization Between Enterprises and the Cloud

Actors: **cloud-subscriber**, **cloud-subscriber-administrator**, **cloud-provider**

Goals: The **cloud-subscriber** requires changes to user credentials in the enterprise's identity provider system to be automatically communicated to the corresponding infrastructure in the **cloud-provider**'s system to ensure the integrity of access and conformance to enterprise policies are maintained in near real time. This is an extension and optimization of the use case for **User Account Provisioning**.

Assumption: The **cloud-subscriber** has well defined policies and capabilities for identity and access management for its enterprise IT applications and data objects. The **cloud-subscriber** has enterprise infrastructure to support the export of user account identity and credential data. The **cloud-provider** has identity provider capabilities and has provided an interface (Web browser-based user interface or an API set) to accept **cloud-subscriber**'s input and/or upload of **cloud-subscriber-user** identity data for account synchronization. The **cloud-provider**'s identity provider capabilities have been setup to communicate securely with the **cloud-provider**'s identity management interface (APIs).

Success Scenario (IaaS): **Steps:** The **cloud-subscriber-administrator** creates/schedules a repeatable job to monitor changes to the enterprise's identity provider store, and configures the policies to synchronize the changes to the **cloud-provider**'s identity management interface (APIs). The scheduled job monitors changes in user identity and credential data, and bulk processes updates to the **cloud-provider**'s identity management sub-system in near real time, thus keeping the identity and credential data in-sync.

Failure Condition/Failure Handling: The **cloud-subscriber-user** accesses the cloud application/service/data in-between of the credential synchronization and breaks integrity of access and conformance to enterprise policy.

Credit:

5.5 eDiscovery

Actors: **cloud-subscriber**, **cloud-provider**, **legal-representative, transport-agent**

Goals: To maintain data objects and their metadata, which are stored and processed in a cloud, so that the data provenance can be known, and to provide data to an authorized **legal-representative** on request. The **cloud-provider** must be able to collect a snapshot of data about the **cloud-subscriber.**

Assumptions: The **legal-representative** has obtained authority from a court to have the **cloud-provider** locate and preserve information of interest.

Success Scenario (ediscovery, IaaS): An authorized **legal-representative** formally requests that a **cloud-provider** disclose information stored on behalf of a **cloud-subscriber**. The **cloud-provider** maintains logs that allow the **cloud-provider** to indicate the provenance of data in the **cloud-provider's** infrastructure that belongs to a specific **cloud-subscriber**. In response to the request, the **cloud-provider** creates a snapshot of the relevant data stored on behalf of the specified **cloud-subscriber**, including data regarding active virtual machines or other processing elements that the **cloud-subscriber** uses or if available, has used. The **cloud-provider** conveys the requested data to the **legal-representative** by an appropriate means (e.g., by **transport-agent** if the data is large). The **legal-representative** may be required to compensate the **cloud-provider** for the costs of providing the service.

Failure Condition: The **cloud-provider** fails to execute the request at all or in part.

Failure Handling: The **legal-representative** must confront the **cloud-provider** via the court system for resolution.

Credit: SNIA has a brief description in its draft use cases [SNIA].

5.6 Security Monitoring

Actors: **cloud-subscriber**, **cloud-provider**

Goals: Conduct ongoing automated monitoring of the **cloud-provider** infrastructure to demonstrate compliance with **cloud-subscriber** security policies and auditing requirements.

Assumption: The **cloud-subscriber** has well defined policies and auditing requirements for its IT infrastructure. Security Content Automation Protocol (SCAP) validated security tools are deployed within the infrastructure to perform monitoring and compliance reporting. The **cloud-subscriber** policies and auditing requirements are expressed in a standard format suitable for automatic processing. The **Cloud-subscriber** may require **cloud-providers** to demonstrate compliance to multiple regulations (e.g., HIPAA, PCI, SOX, FISMA, etc.). The degree of monitoring incumbent upon the **cloud-provider** may vary based on the cloud computing service model in use and the SLA. The regulatory controls required by a client may be derived from a security control framework such as NIST 800-53, ISO/IEC 27002, or the Cloud Security Control

Matrix from the Cloud Security Alliance (being moved into ISO/IEC 27000) to enable a standard control set to compare and monitor security in cloud environments.

Success Scenario 1 (Express Policy and Check Mechanisms, IaaS): **Cloud-subscriber** attempts to convey security monitoring requirements to the **cloud-provider** using standard formats (e.g., SCAP). These requirements are expressed as machine-readable policy documents that describe the required configuration settings, vulnerability and malware detection components, and system patch state. The **cloud-provider** acknowledges successful receipt of the policy content.

Failure Conditions 1: TBD

Failure Handling 1: TBD

Success Scenario 2 (Assess Cloud Environment, IaaS): Cloud-provider continuously monitors cloud components under their purview and demonstrates compliance to the designated policy through the presentation of standardized assessment results to the **cloud-subscriber**. If the **cloud-provider** fails to deliver evidence of compliance within the timeout period, the **cloud-subscriber** may consider an alternate provider or attempt to resubmit the request. Allocation of workload to a **cloud-provider** is contingent upon the ability of the provider to satisfy the **cloud-subscriber** security requirements on an ongoing basis. The failure of a **cloud-provider** to maintain compliance may trigger the migration of the workload to an alternate provider.

Failure Conditions 2: The requested action or process performed at one or more of the *N* **cloud-provider**s fails, is non-responsive, or returns incorrect or incomplete results to the **cloud-subscriber**.

Failure Handling 2: **Cloud-subscriber** can reinitiate the requested action, attempt to mediate discrepancy with the **cloud-provider**, or consider performing the action with an alternative **cloud-provider**.

Credit: TBD

Actors: Cloud-provider, Cloud-subscriber, Country-CERT

Goals: Conduct security monitoring to detect, handle and coordinate incident response between Cloud-subscribers and Cloud-providers, between Cloud-providers, and between country-level CERTs (Country-CERT).

Assumptions: Security Content Automation Protocol (**SCAP**) validated security tools are deployed within the infrastructure to perform monitoring and compliance reporting as well as to detect incidents. Once an incident is detected, the Incident Object Description and Exchange Format (**IODEF**) **[RFC5070]** is used to provide a standard format for the incident investigation over the lifecycle of that incident. Real-time Inter-network Defense (**RID**) **[RFC6045]** is used to communicate the incident information in an IODEF format between entities.

Success Scenario 1: A **Cloud-provider** detects a possible incident via information provided by the use of SCAP in the cloud environment that was able to determine a threat may have been realized (Vulnerability is known via CVE, OVAL results show the configuration would allow for the exploit to be successful, event logs indicate an attempt was made to exploit the vulnerability). The **Cloud-provider** takes the known information about the incident and formats it into IODEF.

The **Cloud-provider** investigates the incident and determines that further research is needed, then sends the IODEF document encapsulated in the RID wrapper to another **Cloud-provider** (or Country-CERT) for further investigation. The second **Cloud-provider** finds the source of the incident and mitigates the traffic. The second **Cloud-provider** communicates the result back to the first **Cloud-provider**. The first **Cloud-provider** may then send a report using RID to the affected client including all of the actions taken to resolve the issue. The first **Cloud-provider** may also send a report using RID to the **Country-CERT** to raise awareness about this attack type.

Failure Conditions 1:

Failure Handling 1:

5.7 Sharing of access to data in a cloud

Actors: **cloud-subscriber**, **unidentified-user, cloud-provider**

Goals: A **cloud-subscriber** makes access to objects stored in a **cloud-provider** selectively available to other **cloud-subscribers** and **unidentified-users**.

Assumptions: The **cloud-provider** provides an Access Control List (ACL) for each data object and for each data object container. An ACL contains a set of ACL entries, each of which lists a set of permitted access modes (*e.g*., read, write, delete, append, truncate, traverse) and the identities of a set of **cloud-subscriber**s to which the modes apply. The **unidentified-user** is a pseudo-**cloud-subscriber** for which access rights are specified. The ACL for a new object-or-container is initialized with a default value that a **cloud-subscriber** can set. A **cloud-subscriber** has administrative access to the ACLs of a set of data objects.

Success Scenario (change-ACL, IaaS, PaaS): A **cloud-subscriber** who owns objects sends a request to a **cloud-provider** to change the ACL for one or more of those objects. The request specifies the object identifier for each object's access modes that should be affected. The change may be the addition or deletion or edit of an existing ACL entry. After the request has been processed, object access requests from the specified **cloud-subscriber**s and **unidentified-users** will be checked in accordance with the new ACL by the **cloud-provider**.

Failure Conditions: (1) a **cloud-subscriber** or **unidentified-user** attempts to modify the ACL (in order to give others access to an object) although the **cloud-subscriber** or **unidentified-user** does not active permission to do so.

Failure Handling: For (1), the data object's owner with the correct permissions will need to make the ACL modification request to the **cloud-provider**.

Requirements File: NA

Credit: ACLs have been included in many systems and specifications, including POSIX.1e.

6. Future Use Cases Candidates

The following use cases are described in enough detail to be included in this report, but have not yet been placed into the above hierarchy and could be expanded to include several related use case scenarios. In some of the use cases given below, separation into distinct sub-cases is implied but has not yet been realized. Incorporation of these use cases also implies and requires additional use cases to be added in complementary sections, as noted below.

Further expansion and integration of the SAJACC use case listing into a more comprehensive organization that takes into account the output of other NIST Cloud Computing groups is discussed in a subsequent section.

6.1 Cloud Management Broker

Actors: **cloud-subscriber, cloud-user, cloud-provider-1, …cloud-provider-n, cloud-management-broker**

Goals: Provide a **cloud-user** a unified and enhanced management interface to multiple **cloud-providers**. The essential features of a **cloud-management-broker** are a unified interface, federated cloud-subscriber credentials for multiple **cloud-provider**s and federated access to multiple **cloud-provider** programming interfaces.

Assumptions: **Cloud-management-broker** services can be delivered in many forms, including as a standalone service, or a set of capabilities within a **cloud-provider**. In cases where a **cloud-provider** business entity also functions as a **cloud-management-broker**, its **cloud-management-broker** aspect is regarded as an entirely separate use case actor**. Cloud-management-broker** services can be also executed by agency custom code. The modal case is assumed to be a third-party service provider independent of and capable of addressing multiple **cloud-providers**.

Success Scenario 1 (generic base, IaaS, PaaS, SaaS): A **cloud-user** wishes to carry out an action on **cloud-provider-1** using a federated interface, with no direct knowledge of **cloud-provider-1** commands or interfaces. A **cloud-management-broker** offers the **cloud-user** a federated interface to multiple **cloud-providers** through a human user interface, an application programming interface or both. The **cloud-user** selects desired **cloud-provider-1** resources, action and action parameters using the **cloud-management-broker** interface. The **cloud-management-broker** collects and marshals the selected action and parameters from the **cloud-user's** selection and issues the desired command to **cloud-provider-1** using **cloud-provider-1** native interface.

Failure Conditions 1: (1) The **cloud-user** command fails at the **cloud-management-broker** because of misconfiguration or incorrect **cloud-management-broker** operation; (2) The **cloud-user** command fails at the target **cloud-provider** due to improper API call or incorrect cloud-provider operation.

Failure Handling 1: **Cloud-management-broker** notifies **cloud-user** of event with diagnostic information and offers retry opportunity. **Cloud-management-broker** notifies its own operational staff and monitoring services to update its own and cloud-provider availability information.

Note that the base scenario failure conditions and handling apply to all scenarios in this use case.

<u>Management Scenarios</u>

Success Scenario 2 (extended management case – Open An Account, IaaS, PaaS, SaaS): A **cloud-subscriber** has opened an account with a **cloud-provider-1** as detailed in the extended management use case and now wishes to manage **cloud-provider-1** using the **cloud-management-broker.** The **cloud-user** registers the **cloud-provider-1** account with the **cloud-management-broker** programming or human interface, and provides sufficient **cloud-provider-1** credentials for the **cloud-management-broker** to address the **cloud-provider-1** native interface. The **cloud-subscriber** may optionally enter descriptive information, spending or other usage limits and metrics to the **cloud-management-broker** to place management limitations on **cloud-provider-1** usage. The **cloud-management-broker** uses **cloud-provider-1's** native interface to validate account and credential information, notifies the **cloud-subscriber** and includes **cloud-provider-1** in its action and metrics interfaces.

Failure Conditions 2 (Base Plus): The **cloud-provider-1** credentials provided by the **cloud-user** are rejected by **cloud-provider-1.**

Failure Handling 2 (Base Plus): **Cloud-management-broker** notifies **cloud-user** of event with diagnostic information and offers opportunity to retry or replace credentials.

Success Scenario 3 (**manage-cloud-user, IaaS, PaaS, SaaS**). A **cloud-subscriber** has registered a **cloud-provider-1** account with a **cloud-management-broker** and wishes to selectively grant and manage their **cloud-users** access to **cloud-provider-1** information, resources and operations. The **cloud-subscriber** uses the **cloud-management-broker** interface to define **cloud-users** and aggregate groupings of **cloud-users** with related resource utilization limits. The **cloud-subscriber** uses the **cloud–management-broker** interface to selectively grant the appropriate **cloud-users** access to specified **cloud-provider-1** information, resources and operations by means of a **cloud-management-broker** access control framework. The **cloud-user** authenticates with the **cloud-management-broker**, accesses an interface presenting permitted information, operations and resources for **cloud-provider-1**, and executes tasks within their scope of permission. The **cloud-management-broker** tracks and reports on **cloud-provider-1** resource utilization for users and aggregates of users, effectively multiplexing the cloud-subscriber account over multiple cloud-users.

Failure Conditions 3 (Base Plus): **Cloud-management-broker** is unable to access **cloud-provider-1** utilization information for a period of time.

Failure Handling 3 (Base Plus): **Cloud-management-broker** notifies **cloud-user** of event with diagnostic information detailing the period of utilization information outage.

Success Scenario 4 (included management case – Close an Account, **IaaS, PaaS, SaaS**): A **cloud –subscriber** has previously registered a **cloud-provider-1** account with **cloud-management-broker** as detailed in **Success Scenario 2 (extended management case – "**Open An Account**")** and now wishes to close the account with **cloud-provider-1**. The **cloud-subscriber** uses the **cloud-management-broker** interface to issue **cloud-provider-1** a close account command. The **cloud-management-broker** accesses **cloud-provider-1** using **cloud-subscriber** credentials, marshals parameters from the **cloud-subscriber**, and issues **cloud-provider-1** commands implementing the management use case "Close An Account". The **cloud-management-broker** delivers **cloud-subscriber** all consequent **cloud-provider-1** messages including non-repudiation information.

Failure Conditions 4 (Base Plus): **Cloud-provider-1** does not offer an account close interface.

Failure Handling 4 (Base Plus): **Cloud-management-broker** does not make close account command available to **cloud-user**.

Success Scenario 5 (included management case – Terminate an Account, **IaaS, PaaS, SaaS**): A **cloud–subscriber** has previously registered a **cloud-provider-1** account with **cloud-management-broker** as detailed in **Success Scenario 2 (extended management case – "Open An Account")**. **Cloud-provider-1** now wishes to terminate an account as detailed in included management Use case "Terminate an Account." The **cloud-management-broker** regularly polls all registered **cloud-provider**s for status events, detects the account freeze or termination notification per included management use case and conveys the notification to the **cloud-subscriber** through the **cloud-management-broker** interface. The **cloud-subscriber** optionally communicates directly with **cloud-provider-1** to reinstate the terminated account if desired.

Failure Conditions 5 (Base Plus): The **cloud-provider-1** freeze or notification is not provided by cloud**-provider-1** programming interface and is not seen by **cloud-management-broker.**

Failure Handling 5 (Base Plus): **Cloud-subscriber** receives **cloud-provider-1** freeze or termination notification directly from **cloud-provider-1** and proceeds per included management case.

Success Scenario 6 (included management cases – Copy Data Objects into a Cloud/Network , Copy Data Objects out of a Cloud/Network to Cloud User, Erase Data Objects In a Cloud, IaaS): A **cloud-user** wishes to copy data objects into, out of, or erase objects on a **cloud-provider-1** cloud as detailed in the included management cases. The **cloud-user** accesses a **cloud-management-broker** interface to view their **cloud-provider-1** storage structures and issues **cloud-management-broker** commands to copy data objects into **cloud-provider-1**, copy out of **cloud-provider-1** or erase objects from **cloud-provider-1**. The **cloud-management-broker** uses **cloud-provider-1**'s native interface to issue commands to effect the operation specified in the included use case and notifies **cloud-user** of the result. Note that the "copy" management scenarios for **unidentified user**s or **transport-agent** data transfer are not covered by the **cloud-management-broker**, as unauthenticated access and physical transport are inappropriate for brokerage.

Failure Conditions 6 (Base Plus): No conditions for scenario beyond base

Failure Handling 6 (Base Plus): TBD

Success Scenario 7 (included management cases VM Control: Allocate VM Instance, VM Control: Manage Virtual Machine Instance, IaaS): A **cloud-user** wishes to allocate or manage virtual machine instances as detailed in the included management cases. The **cloud-user** accesses a **cloud-management-broker** interface to view **cloud-provider-1** virtual machine images and instances, and issues **cloud-management-broker** commands to allocate or manage selected instances as specified in the included use case. The **cloud-management-broker** uses **cloud-provider-1's** native interface to issue commands to effect the operation specified in the included use case and notifies **cloud-user** of the result.

Failure Conditions 7 (Base Plus): No conditions for scenario beyond base.

Failure Handling 7 (Base Plus): TBD

Success Scenario 8 (included management case Monitor Infrastructure [DOES NOT EXIST IN MANAGEMENT SECTION YET], IaaS): A **cloud-user** wishes to monitor and respond to changes in infrastructure services provided by **cloud-provider-1**. **Cloud-provider-1** is previously registered by **cloud-subscriber** with the **cloud-management-broker** per **Success Scenario 2 (extended management case – Open An Account). Cloud-user** has sufficient permissions to set thresholds, view reports and receive alerts per **Success Scenario 3** (**manage-cloud-user**). The **cloud-management-broker** assembles **cloud-provider-1** performance and availability information using native **cloud-provider-1** interfaces, and aggregates the information in its internal reporting and alerting framework. The **cloud-user** uses the **cloud-management-broker's** interface to set alerting thresholds on **cloud-provider-1** infrastructure components, view reports on cloud-provider-1 infrastructure component performance and availability, and receive alerts on **cloud-provider-1** infrastructure components when performance triggers the alerting thresholds.

Failure Conditions 8 (Base Plus): (1) **Cloud-management-broker** is unable to access **cloud-provider-1** monitoring information for a period of time; (2) **Cloud-management-broker's** internal monitoring processes are unavailable for a period of time

Failure Handling 8 (Base Plus): (1) The **cloud-management-broker** treats unavailable **cloud-provider-1** monitoring information as an alert and transmitted to **cloud-users** registered for alerts on **cloud-provider-1**; (2) Unavailable **cloud-management-broker** monitoring services are treated as an alert and transmitted to **cloud-subscriber** and **cloud-users** registered for alerts, either through **cloud-management-broker** alerting system if functional, by email if alerting is not functional, or after the fact if entire **cloud-management-broker** is inoperative.

<u>Interoperability Scenarios</u>

Success Scenario 9: (Migrate Data Objects Between Clouds, IaaS): A **cloud-user** wishes copy or move a data object from **cloud-provider-1** to **cloud-provider-2**. The **cloud-user** accesses a **cloud-management-broker** interface to access both **cloud-provider-1** and **cloud-provider-2** data objects and containers in order to select source data objects from **cloud-provider-1**, destination data objects from **cloud-provider-2** and the desired mode of migration. The copy mode leaves the source object intact after migration; the move operation provides transactional erasure of the source object. For each selected **cloud-provider-1** data object, the **cloud-management-broker** uses **cloud-provider-1's** native interface to issue commands to access the object, and **cloud-provider-2's** native interface to create a new object with identical content in the location indicated in the copy or move command, verifies the integrity of the new object, and notifies **cloud-user** of the result. In the move mode the **cloud-management-broker** then issues a native **cloud-provider-1** command to erase the source object.

Failure Conditions 9 (Base Plus): (1) A namespace collision between the desired destination location on **cloud-provider-2** and the specified destination identifier occurs; (2) One or several of a series of copy or move operations fail but some succeed; (3) The transfer portion of a move transaction succeeds, but the erasure portion fails.

Failure Handling 9 (Base Plus): (1) The **cloud-management-broker** detects the namespace collision before initiating the copy operation and notifies the **cloud-user** with option to overwrite, skip or, in the case of a series of commands, skip all. (2) On first fail the **cloud-management-broker** notifies the **cloud-user** of the failure and offers the option to retry, skip or abort. (3) On failed erasure the **cloud-management-broker** notifies the **cloud-user** and offers the option to roll back the transaction, which erases the destination object, or to retry.

Success Scenario 10: (included interoperability cases – Cloud Burst From Cloud to Cloud, IaaS): A **cloud-user** configures rules with the **cloud-management-broker** governing how service requests are allocated over a pool of registered providers, **cloud-provider-1** thru **cloud-provider-n**. The rules establish a precedence of providers, a way to query and respond to reported provider load, and metrics to allocate load over providers in the pool. **Cloud-user** issues a command to **cloud-management-broker** to perform a Virtual Machine operation per **Success Scenario 7 (included management cases VM Control)**, but without identifying a target **cloud-provider**. **Cloud-management-broker** evaluates the command against allocation rules, dynamically selects optimal **cloud-provider(s)** from the **cloud-subscriber's** registered pool, and apportions requests among the **cloud-provider-1** to **cloud-provider-n** using native **cloud-provider** interface to address each involved **cloud-provider**.

Failure Conditions 10 (Base Plus): (1) **Cloud-management-broker** unable to locate an operating cloud-provider based on rules; (2) Command failure on individual **cloud-provider**.

Failure Handling 10 (Base Plus): (1) **Cloud-management-broker** notifies **cloud-subscriber** of event with diagnostic information and offers retry opportunity**;** (2) **Cloud-management-broker** accesses allocation rules to select fallback **cloud-provider** to replace failing provider, and notifies **cloud-subscriber**. In cases where no fallback is available, **cloud-management-broker** notifies subscriber.

Success Scenario 11: (included interoperability cases – Cloud Burst From Data Center to Cloud, IaaS):

Additional Assumptions: Cloud-management-broker has mechanisms for registering and monitoring Data Center load metrics as if the Data Center were a cloud provider. This can be generically implemented with Data Center private cloud software or could be an additional feature of the cloud-management-broker programming interface.

Extends **Success Scenario 10: (Cloud Burst From Cloud to Cloud)** by configuring rules to designate agency Data Center as a member of the pool of cloud providers, and bursting to be allocated over data center and **cloud-provider-1** thru **cloud-provider-n** based on load and rule priority settings.

Failure Conditions (Base Plus Scenario 10): Data center becomes unavailable to **cloud-management-broker** interface

Failure Handling (Base Plus Scenario 10): The **cloud-management-broker** notifies the **cloud-subscriber** of the unavailable data center as if it were an unavailable cloud-provider. See Failure Handling (1) for **Success Scenario 8 (Monitor Infrastructure)**

Success Scenario 12 (included interoperability case Migrate a Queuing-Based Application, IaaS):

Additional Assumptions: An industry agreed minimum set of agreed attributes for queues and messages, and minimum set of queue operations. Java message service (JMS) is an example of an existing cross-implementation specification for queues, topics and messages.

A **cloud-user** wishes to migrate a queue and its current contents from **cloud-provider-1** to **cloud-provider-2** as detailed in the included management case. The **cloud-user** views **cloud-provider-1** queues and messages using the **cloud-management-broker** interface, and issues the **cloud-management-broker** commands to stop and then to migrate queues and messages to **cloud-provider-2** as specified in the included use case. The **cloud-management-broker** uses both **cloud-provider-1** and **cloud provider 2's** native interface to issue commands to effect the migration as detailed in the included use case, and notifies **cloud-user** of the result.

Failure Conditions (Base Plus): (1) Queue stop operation fails on **cloud-provider-1**; (2) Queue creation and/or message transfer fails on **cloud-provider-2**

Failure Handling (Base Plus): (1) **Cloud-management-broker** aborts entire queue migration and notifies **cloud-user** with error message; (2) **Cloud-management-broker** restarts queue on **cloud-provider-1**, aborts queue migration and notifies **cloud-user** with error message.

Success Scenario 13 (included management case Migrate Fully-Stopped) VMs from one provider to another, IaaS)

Additional Assumptions: Cloud-provider-1 supports and exposes an interface to prepare instances and/or machine images for migration. **Cloud-provider-2** supports and exposes an interface to receive prepared instances and convert them to a native operating instance.

A **cloud-user** wishes to migrate a fully-stopped VM instance or machine image from **cloud-provider-1** to **cloud-provider-2** as detailed in the included management case. The **cloud-user** accesses a **cloud-management-broker** interface and views **cloud-provider-1** instances and/or machine images, then issues **cloud-management-broker** commands to stop and then to migrate the selected instances or images as specified in the included use case. The **cloud-management-broker** uses both **cloud-provider-1** and **cloud provider 2's** native interface to issue commands to effect the migration, and notifies **cloud-user** of the result. The **cloud-management-broker** may be responsible for transferring the prepared static representation of the **cloud-provider-1** image to **cloud-provider-2**, but the mechanics of preparing the static representation and subsequently translating it into **cloud-provider-2** format are delegated to the respective cloud provider interfaces per included use case.

Failure Conditions 13 (Base Plus): **Cloud-management-broker** unable to transfer prepared static representation of **cloud-provider**-1 image to **cloud-provider**-2 because of internal or communication failure.

Failure Handling 13 (Base Plus): **Cloud-management-broker** offers **cloud-user** opportunity to retry or abort migration.

<u>**Extended Management Scenarios**</u>

Success Scenario 14 (Extend Infrastructure Instance Management Capabilities, IaaS): A **cloud-user** wishes to perform virtual machine instance management tasks that are not supported by one or all of **cloud-provider-1** through **cloud-provider-n**. Tasks include installing applications or services, creating or managing users, starting and stopping instance services or any other operation that can be performed on a running virtual machine instance but is not supported by the **cloud-provider's** native interface. The **cloud-management-broker** provides the **cloud-user** a management agent to install in each **cloud-provider-1-n** instances, and an interface to issue the **cloud-management-broker** extended commands. **Cloud-user** selects a task, for example installing an application or creating a user, and a collection of instances to receive the task. The **cloud-management-broker** communicates with each instance management agent and issues commands to affect the task that is carried out on the target instances by the locally running agent installed on each virtual machine instance for **cloud-provider**. The **cloud-management-broker** interface represents the state of each command to the **cloud-user**.

Failure Conditions 14 (Base Plus): **Cloud-management-broker** agent fails to execute or reports an error on target instance.

Failure Handling 14 (Base Plus): **Cloud-management-broker** notifies **cloud-user** with error specifics or inability to contact. Transient errors can be retried.

Success Scenario 15 (Assemble and Manage Infrastructure Components as a Platform – IaaS, PaaS): A **cloud-user** wishes to define, assemble and manage a collection of infrastructure components as a coherent multi-tiered platform, including but not limited to virtual machine instances or images, storage, load balancers, and databases. Using a **cloud-management-broker** interface, the **cloud-user** selects **cloud-provider** instances or images and assembles them into tiers, such as application tiers which implement an application layer, load balancing tiers which implement load balancers, or database tiers which implement replicated or clustered databases. The **cloud-user** accesses a **cloud-management-broker** interface and assembles the tiers into a logical platform, specifying scaling metrics for each tier, connections between tiers, data load processes and sources, backup processes and all required interdependencies. The **cloud-user** starts the unified platform using the **cloud-management-broker** interface, views metrics on its various components, receives alerts and alarms on failure or scaling events, views backups, and stops components of the platform or platform as a whole. The **cloud-management-broker** uses native interfaces to **cloud-provider-1 thru n** implementing the platform components, and issues the native commands corresponding to **cloud-user** requests, monitors the **cloud-provider** components, and issues scaling commands and alerts according to **cloud-provider** metrics. Components within a platform assembly may initially be constrained to resources from a single **cloud-provider**, but future cross-provider platform assembly and operation scenarios are possible.

Failure Conditions 15 (Base Plus): (1) **Cloud-management-broker** is unable to fully start or stop the assembled platform because of a processing tier failure; (2) Processing failure within an application tier renders tier inoperative or degraded.

Failure Handling 15 (Base Plus): (1) **Cloud-management-broker** notifies **cloud-user** with error specifics from failing tier component. Transient errors can be retried; (2) **Cloud-management-broker** detects failure of component or tier and restarts tier components according to tier scaling rules.

Credit: RightScale, Enstratus, CompatibleOne.

6.2 Transfer of ownership of data within a cloud

Actors: **cloud-subscriber-1**, **cloud-subscriber-2**, **cloud-provider**

Goals: **Cloud-subscriber-1** transfers the ownership of some data objects from **cloud-subscriber-1** to **cloud-subscriber-2** in a **cloud-provider**.

Assumptions: Cloud-subscriber-1 owns a set of data objects stored with a **cloud-provider**

Success Scenario (transfer of ownership, IaaS): **Cloud-subscriber-1** sends a change-ownership request to the **cloud-provider**. The change-ownership request identifies the objects to be affected, the identity of the **cloud-subscriber** to receive the ownership of the objects (**cloud-subscriber-2**), and the time the change should occur. Either **cloud-subscriber-1** or the **cloud-provider** sends a request to **cloud-subscriber-2** offering the ownership. **Cloud-subscriber-2** accepts or declines the offer. If **cloud-subscriber-2** accepts the offer, immediately after the specified time, the **cloud-provider** changes the ownership metadata for the specified objects and fees associated these objects stop accruing to **cloud-subscriber-1** and begin accruing to **cloud-subscriber-2**.

Failure Conditions: (1) **Cloud-subscriber-1** is not authorized to change ownership; (2) **cloud-subscriber-2** does not respond to the transfer of ownership request; (3) **cloud-provider** does not have access to the data objects.

Failure Handling: **Cloud-provider** notifies **cloud-subscriber-1** that the transfer of ownership request has failed and provides description of why the transfer failed.

Requirements File: The change of ownership request, acceptance or rejection, is logged by the **cloud-provider**. The change of ownership transaction is supported by cryptographic mechanisms that allow for mutual authentication and non-repudiation.

Credit: TBD

6.3 Fault-Tolerant Cloud Group

Actors: **cloud-subscriber, cloud-provider-1, cloud-provider-2, cloud-provider-n**

Goals: Synthesize a highly reliable service using the facilities of multiple **cloud-providers**.

Assumptions: Assume that a **cloud-subscriber** has already opened accounts with *N* **cloud-providers** (See Use case "Open An Account"). We also assume that when comparisons of data or output results from the N cloud-providers are made, a majority of the data or results will be found to be equivalent. Also, the metadata about data objects includes time stamps or sequence numbers.

Success Scenario 1 (write data, IaaS, PaaS): The **cloud-subscriber** attempts to copy a data object onto all N of the **cloud-providers** using the data object APIs that each **cloud-provider** publishes (See Use Case "Copy Data Objects Into A Cloud"). Each **cloud-provider** returns a message indicating whether or not the copy operation succeeded. The **cloud-subscriber** records the number of successes M. If $M < N$, the **cloud-subscriber** may re-issue the request or evaluate whether or not the data has been stored with sufficient redundancy. If not, the **cloud-subscriber** may optionally open accounts with new **cloud-providers**.

Success Scenario 2 (read data, IaaS, PaaS): Assume the **cloud-subscriber** issues a number *K* of concurrent object read requests using the data object APIs that each **cloud-provider** publishes. The **cloud-subscriber** will choose *K* to be large enough so that at least one of the responses from the responding **cloud-provider**s will contain data from the object's most recent update. The **cloud-subscriber** compares responses from the responding **cloud-providers**, and chooses the response representing the latest version of the object.

Success Scenario 3 (redundant batch jobs, IaaS, PaaS): The **cloud-subscriber** starts a processing job on each of the N **cloud-provider**s (e.g., See Use Case "VM Control: Manage Virtual Machine Instances"). Each **cloud-provider** runs exactly the same job, on the same input data, and produces output data. The **cloud-subscriber** retrieves the output data from the first-completing **cloud-provider**, checksums it, and then checksums the output subsequently returning **cloud-provider**s, comparing each for equality. If any of the equality checks fail, the **cloud-subscriber** can rerun the job, perhaps allocating it onto a different set of **cloud-provider**s, or simply take a majority vote and consider that the result.

Success Scenario 4 (state machine replication, IaaS, PaaS): The **cloud-subscriber** starts a long-running server process in each of the *N* **cloud-providers**. Iteratively, the **cloud-subscriber** sends a service request to each server process in the *N* **cloud-providers**, receives each server's results, and compares the results. If the comparisons do not show equality, the **cloud-subscriber** re-initializes servers that are determined to have failed by perhaps migrating to new **cloud-providers**. If a server has failed to respond to requests for a timeout period, the **cloud-subscriber** reinitializes the server, bringing it up to the state of the others.

Failure Conditions: The requested action or process performed at one or more of the *N* **cloud-provider**s fails or produces incorrect returning data to **cloud-subscriber**.

Failure Handling: **Cloud-subscriber** either reinitiates the requested action, or considers performing the action with new **cloud-provider**(s).

Requirements File: NA

Credit: Note: there is a lot of literature on how to implement replication in network services using protocols such as two-phase-commit or quorum-consensus or timestamps or transactions; this is just a sketch. One good source of information on how to compare results (termed "voting") can be found in the *n*-version programming literature.

7. Examples of Validation Tests Conducted Against the SAJACC Use Cases

The use case scenarios given above have been designed to facilitate demonstration and testing of cloud products, cloud API usage, and of the applicability of standards and standards-based software approaches in real-world settings. While these have not yet risen to the level of full conformance testing or of conformity assessment, the SAJACC use cases do already permit such demonstrations and are a step along the road to a full validation program. In the basic diagram of the SAJACC process illustrated in Figure 1 in Section 1 of this report, this type of demonstration corresponds to step 4 of the process.

Several of the SAJACC use cases have already attracted community-based demonstration examples. Other examples were written by a NIST contractor, Jin Tong, and refined iteratively with input from the SAJACC working group. The output of these exercises spanned most of the use cases in the Cloud Management category (Section 3 of this report) and several from the Cloud Interoperability category (Section 4). Community input during these example validation tests was received from one commercial vendor (Microsoft Corporation for the Azure product) and two different standards organizations (SNIA for the CDMI reference implementation, and OGF for some OCCI-related examples running against an OpenNebula instance).

While not yet comprehensive, these examples have been sufficient to demonstrate the potential usefulness of a larger-scale effort to organize, gather or conduct, document and validate demonstrations of this nature. In addition, several of the NIST-prepared examples and community contributions resulted in downloadable test code published on the NIST Cloud Computing TWiki that remains available for those who would like to reproduce or to extend these results on their own. In the "Conclusions" section below, we include a recommendation for NIST to develop and support a more comprehensive method to carry out and/or collect such demonstrations, with input from the SAJACC working group and other relevant NIST Cloud Computing working groups as appropriate.

7.1 Examples of SAJACC Use Case Reports

The first example report below for SAJACC use case 3.4 "Copy Data Objects Into A Cloud" was presented on Feb. 15, 2011 in teleconference number 6, carried out the Amazon S3 protocol using AWS services and also tested against a NIST-hosted instance of Eucalyptus. A later similar demonstration of this use case was conducted with the Azure cloud product. Additional example reports are included for use cases 3.7 "Allocate VM Instance", 4.1 "Copy Data Objects Between Cloud Providers" and 5.7 "Sharing Access To Data In A Cloud". A larger collection of reports, code and annotated listings is contained in the SAJACC meeting TWiki pages. Listings of the source code used to prepare these reports are included in Appendix D.

NIST Cloud Computing Use Case Testing Report

3.4 Copy Data Objects Into A Cloud

This test driver implements `Success Scenario 1 (cloud-subscriber-to-network copy, IaaS, PaaS, SaaS)` of Use Case 3.4.

Test Scenario:

- **The cloud-subscriber determines a local file**, `helloworld.txt`, **for copying to the cloud-provider's system**.
- The cloud-subscriber creates a client handle in preparation to issue commands to the cloud-provider's system using pre-acquired credentials.
- **The cloud-subscriber issues a command to create a container**: `S3Bucket [name=test-bucket-sajacc-usecases-3-4675737726,location=US,creationDate=null,owner=null] Metadata={Content-Length=0, storage-class=STANDARD, Content-Type=application/xml}`.
- **The cloud-subscriber issues a command to create an object in the container created, and transfers the local file** `helloworld.txt`**to the cloud-provider's system**.

Verifying test result:

- Download Test Object: S3Object [key=helloworld.txt, bucket=test-bucket-sajacc-usecases-3-4675737726, lastModified=Tue Feb 15 12:38:17 EST 2011, dataInputStream=org.jets3t.service.impl.rest.httpclient.HttpMethodReleaseInputStream@337d0f, Metadata={ETag=8ddd8be4b179a529afa5f2ffae4b9858, Content-Length=13, Last-Modified=Tue Feb 15 12:38:17 EST 2011, md5-hash=8ddd8be4b179a529afa5f2ffae4b9858, Content-Type=application/octet-stream}].
- Download Test Object converted to string as: Hello World!.
- MD5 hash comparison against source file `helloworld.txt` returned `true`.

Cleaning up testing objects: `helloworld.txt`.

Cleaning up testing container: `test-bucket-sajacc-usecases-3-4675737726`.

NIST Cloud Computing Use Case Testing Report

3.7 VM Control: Allocate VM Instance

This test driver implements `Success Scenario 1 ((AllocateVM, IaaS))` of Use Case 3.7 using AWS SDK for Java.

Test Scenario - `Success Scenario`:

- The cloud-subscriber creates a client handle in preparation to issue commands to the cloud-provider's system using pre-acquired credentials.
- **(1) The cloud-subscriber requests a specific pre-defined Virtual Machine image supplied by the cloud-provider (O/S, CPU cores, memory, and security) and launches new VM instances**.
    - Started Instance `{InstanceId: i-49FF098A, ImageId: emi-F68218F3, State: {Code: 0, Name: pending, }, PrivateDnsName: 0.0.0.0, PublicDnsName: 0.0.0.0, StateTransitionReason: NORMAL: -- [], KeyName: uc_test, AmiLaunchIndex: 0, ProductCodes: null, InstanceType: c1.medium, LaunchTime: Tue Mar 22 10:51:55 EDT 2011, Placement: {AvailabilityZone: cluster1, GroupName: null, }, KernelId: eki-6B1B1E12, RamdiskId: eri-45281D7A, Platform: null, Monitoring: {State: false, }, SubnetId: null, VpcId: null, PrivateIpAddress: null, PublicIpAddress: null, StateReason: null, Architecture: null, RootDeviceType: null, RootDeviceName: null, BlockDeviceMappings: null, VirtualizationType: null, InstanceLifecycle: null, SpotInstanceRequestId: null, License: null, ClientToken: null, Tags: null, }`
    - Waiting for the VM instance: i-49FF098A to be ready
    - The VM instance is ready, details: `{InstanceId: i-49FF098A, ImageId: emi-F68218F3, State: {Code: 16, Name: running, }, PrivateDnsName: 172.19.1.34, PublicDnsName: 192.168.2.51, StateTransitionReason: NORMAL: -- [UPDATE], KeyName: uc_test, AmiLaunchIndex: 0, ProductCodes: null, InstanceType: c1.medium, LaunchTime: Tue Mar 22 10:51:55 EDT 2011, Placement: {AvailabilityZone: cluster1, GroupName: null, }, KernelId: eki-6B1B1E12, RamdiskId: eri-45281D7A, Platform: null, Monitoring: {State: false, }, SubnetId: null, VpcId: null, PrivateIpAddress: null, PublicIpAddress: null, StateReason: null, Architecture: null, RootDeviceType: null, RootDeviceName: null, BlockDeviceMappings: null, VirtualizationType: null, InstanceLifecycle: null, SpotInstanceRequestId: null, License: null, ClientToken: null, Tags: null, }`
- **(3) The cloud-subscriber has secure launching and administration of their VM instance**.
    - The cloud-subscriber issues shell command `uname -a; uptime` through SSH session extablished to the VM instance using the private key from key pair:.uc_test
    - Issue command:uname -a; uptime to ubuntu@192.168.2.51
        - Shell command returns: `Linux ip-172-19-1-34 2.6.31-22-generic-pae #73-Ubuntu SMP Fri Feb 11 18:39:01 UTC 2011 i686`

```
GNU/Linux 14:53:44 up 1 min, 0 users, load average: 0.38,
0.11, 0.03
```

Cleaning up: Terminate the running VM instance: `i-49FF098A`.

- Instance terminated.

NIST Cloud Computing Use Case Testing Report

3.7 VM Control: Allocate VM Instance

This test driver implements `Success Scenario 1 ((AllocateVM, IaaS))` of Use Case 3.7 using OCCI API.

Test Scenario - `Success Scenario`:

- **(1) The cloud-subscriber requests a specific pre-defined Virtual Machine image supplied by the cloud-provider (O/S, CPU cores, memory, and security) and launches new VM instances**.
  - Started Instance `50`
  - The VM instance is ready, details: `<COMPUTE href="http://clc:4567/compute/50"><ID>50</ID><CPU>1</CPU><MEMORY>1024</MEMORY><NAME>sajacc-test-vm</NAME><INSTANCE_TYPE>small</INSTANCE_TYPE><STATE>ACTIVE</STATE><DISK id="0"><STORAGE href="http://clc:4567/storage/2" name="ttylinux"/><TYPE>DISK</TYPE><TARGET>hda</TARGET></DISK><NIC><NETWORK href="http://clc:4567/network/4" name="Public network"/><IP>192.168.1.242</IP><MAC>02:00:c0:a8:01:f2</MAC></NIC></COMPUTE>`
- **(3) The cloud-subscriber has secure launching and administration of their VM instance**.
  - The cloud-subscriber issues shell command `uname -a; uptime` through SSH session extablished to the VM instance using the private key from key pair.
  - Issue command:uname -a; uptime to sajacc@192.168.1.242
    - Shell command returns: `Linux sajacc-test-vm 2.6.20 #1 PREEMPT Mon Aug 17 20:32:57 MST 2009 i686 GNU/Linux 01:14:55 up 0 min, load average: 0.00, 0.00, 0.00`

Cleaning up: Terminate the running VM instance: `50`.

- Instance terminated.

NIST Cloud Computing Use Case Testing Report

4.1 Copy Data Objects between Cloud-Providers

This test driver implements `Success Scenario 1 of Use Case 4.1 using Eucalyptus Cloud as **cloud-provider-1** and CDMI as **cloud-provider-2**.

Test Scenario:

- **The cloud-subscriber** authenticates to **cloud-provider-1** (an Eucalyptus/Walrus Interface) and creates object 'test-container-sajacc-usecases-4-1559744109/helloworld.txt'.
- **The cloud-subscriber** authenticates to **cloud-provider-1** (an Eucalyptus VM) and starts an SSH session.
- The cloud-subscriber executes a command on **cloud-provider-1** to download a data object from the S3 service from the same cloud provider through an SSH session.
    - HTTP get (`s3curl`) command is:

      ```
      source ~/test/eucarc; cd ~/test/s3-curl; ./s3curl.pl --id
      $EC2_ACCESS_KEY --key $EC2_SECRET_KEY --get -- -s -v
      $S3_URL/test-container-sajacc-usecases-4-
      1559744109/helloworld.txt > ~/helloworld.txt
      ```

- **The cloud-subscriber** determines to copy `helloworld.txt` to **cloud-provider-2** (a CDMI provider).
- **The cloud-subscriber run a shell command from __cloud-provider-1** (an Eucalyptus VM) to create a container: `test-container-sajacc-usecases-4-1559744109` in **cloud-provider-2** (a CDMI provider).
    - the command used to create container is:

      ```
      curl -i -X PUT http://cdmi-server:9090/cdmi-server/test-
      container-sajacc-usecases-4-1559744109 --header 'Content-
      type:application/vnd.org.snia.cdmi.container+json' --header 'X-
      CDMI-Specification-Version:1.0' --data '{ "metadata":{ } }'
      ```

- **The cloud-subscriber** run a shell command from **cloud-provider-1** (an Eucalyptus VM) to create an object 'helloworld.txt' in the container created.
    - the command used to create object is:

      ```
      curl -i -X PUT http://cdmi-server:9090/cdmi-server/test-
      container-sajacc-usecases-4-1559744109/helloworld.txt --header
      'Content-type:application/vnd.org.snia.cdmi.dataobject+json' --
      header 'X-CDMI-Specification-Version:1.0' --data '{
      "mimetype":"text/plan", "value":"Hello World!"}'
      ```

Verifying test result:

- Download Source Object from **cloud-provider-1(Eucalyptus)**
- Downloaded Source Object converted to string as: Hello World!.
- Download Destination Object from **cloud-provider-2(CDMI)**.
- Downloaded Destination Object converted to string as: Hello World!.
- Compare against source data returned `true`.

Cleaning up testing objects from CDMI: `helloworld.txt`.

Cleaning up testing container from CDMI: `test-container-sajacc-usecases-4-1559744109`.

Cleaning up testing objects from S3: `helloworld.txt`.

Cleaning up testing container from S3: `test-container-sajacc-usecases-4-1559744109`.

NIST Cloud Computing Use Case Testing Report

5.7 Sharing of access to data in a cloud

This test driver implements `Success Scenario 1 A cloud-subscriber who owns objects sends a request to a cloud-provider to change the ACL for one or more of those objects of Use Case 5.7 using S3 interface.

Test Scenario - `Success Scenario 1`:

- The cloud-subscriber issues a command to create a container: `test-bucket-sajacc-usecases-5-71347539391`, and create an object in the container created, and transfers the local file, `helloworld.txt`, to the cloud-provider's system.
- Trying to access the object without "read" (or equivalent) access to the object, with all default ACL setting, `/test-bucket-sajacc-usecases-5-71347539391/helloworld.txt`, got back the following response:
    - response status code = 403
    - response reason phrase = Forbidden
- **The cloud-provider provides an Access Control List (ACL) for each data object and for each data object container. An ACL contains a set of ACL entries, each of which lists a set of permitted access modes (e.g., read, write, delete, append, truncate, traverse) and the identities of a set of cloud-subscribers to which the modes apply**. In this case, the cloud-subscriber updates the ACL of the data object to grant (read) access right to all *authenticated users*.
- As an unauthenticated user, trying to access the object after the cloud-subscriber granted "read" (or equivalent) access to only authenticated, `/test-bucket-sajacc-usecases-5-71347539391/helloworld.txt`, as expected got back the following response (expecting http 403):
    - response status code = 403
    - response reason phrase = Forbidden
- Authenticated as test user 2 (`testuser2`), who has been granted read permission to the object. Downloaded the object (`test-bucket-sajacc-usecases-5-71347539391, helloworld.txt`): S3Object [key=helloworld.txt, bucket=test-bucket-sajacc-usecases-5-71347539391, lastModified=Tue Mar 08 11:54:28 EST 2011, dataInputStream=org.jets3t.service.impl.rest.httpclient.HttpMethodReleaseInputStream@12be1bd, Metadata={ETag=8ddd8be4b179a529afa5f2ffae4b9858, Content-Length=13, Last-Modified=Tue Mar 08 11:54:28 EST 2011, md5-hash=8ddd8be4b179a529afa5f2ffae4b9858, Content-Type=application/octet-stream}]

Cleaning up testing objects: `helloworld.txt`.

Cleaning up testing container: `test-bucket-sajacc-usecases-5-71347539391`.

Editor's Notes:

Need to add two account credentials in the configuration file.

8. Comparison with other NIST Cloud Computing Group Organizational Structures, Roadmaps, and Output

The organization and numbering of the original SAJACC use cases was set early in the process that produced these scenarios, and was retained to permit consistent validation exercises to be conducted according to the procedure outlined in Figure 1 of Section 1 of this report. During the intervening two years, as SAJACC validation was being carried out, a much more complete Reference Architecture and Taxonomy were developed by the corresponding NIST Cloud Computing working group, and extensive other roadmap organizational activities were carried out by the NIST Cloud Computing Standards Roadmap working group.

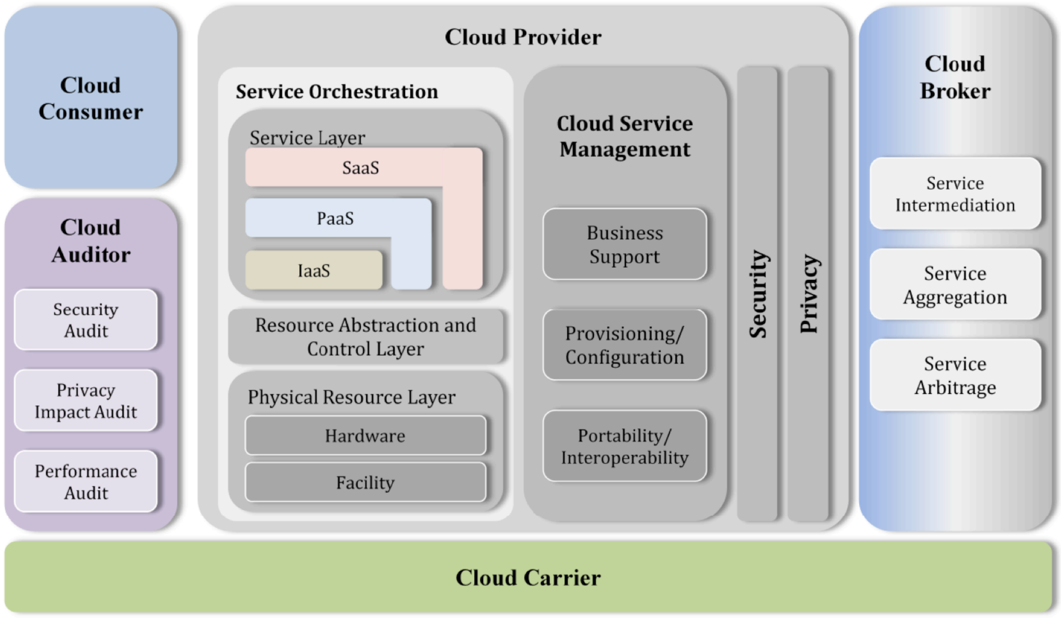


Figure 3: NIST Cloud Computing Reference Architecture

Other, more general Federal Cloud Computing business use cases were developed and documented by the NIST Cloud Computing Business Use Cases group, which were used as input wherever possible in developing the SAJACC technical use cases presented here. The NIST Cloud Computing Security working group has also conducted an extensive survey of the cloud computing security area. This survey resulted in a comprehensive list of cloud computing security impediments and remedies and an associated report.

This section of the current document is devoted to comparison of the organizational structure of the SAJACC use case listing with the output of the above groups. Wherever possible, these have been derived directly from the working group materials and reports as documented on the NIST Cloud Computing web site and TWiki.

In the following sub-sections, the diagrams given can be compared with the original NIST SAJACC working group use case organizational diagram provided in Figure 2 of Section 2 above.

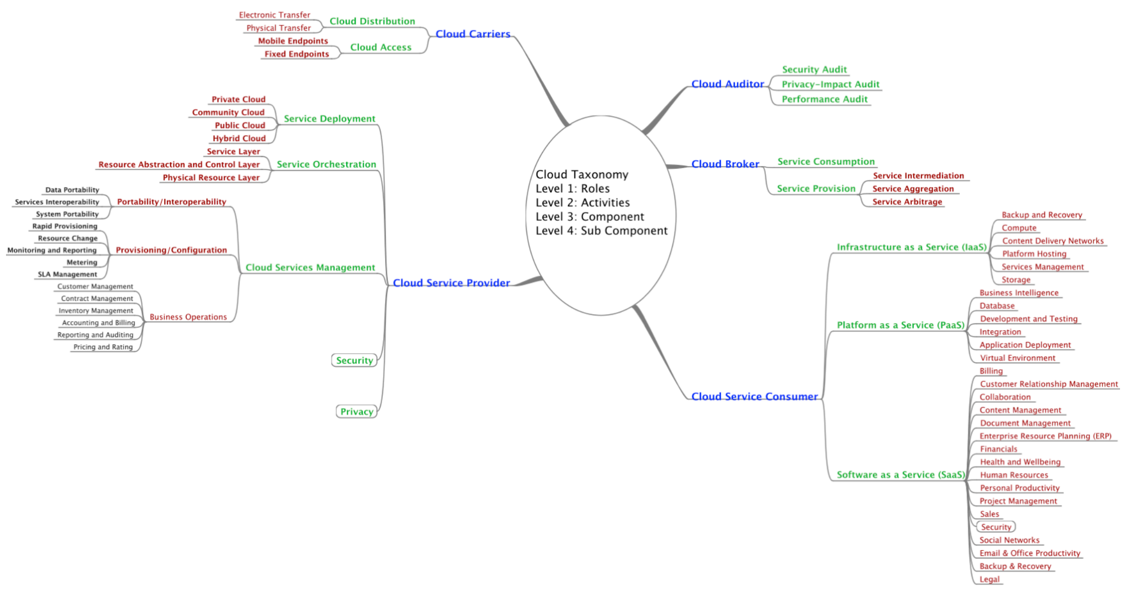


Figure 4: NIST Cloud Computing Reference Architecture Taxonomy (overall view)

8.1 NIST Cloud Computing Reference Architecture and Taxonomy

Figure 3 gives the current NIST Cloud Computing Reference Architecture diagram as contained in NIST Special Publications 500-292 and 500-293(Volumes I-II). The process that was used to develop this architecture also resulted in an extensive taxonomy, which is shown in a high-level overview in Figure 4, and specialized taxonomies for the Cloud Service Agreement and Cloud SLA components, as shown in Figures 5 and 6.

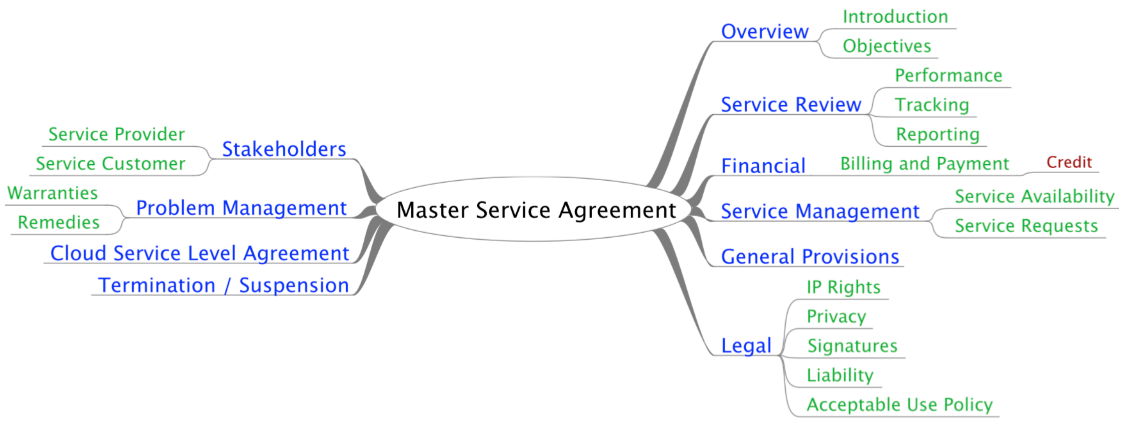


Figure 5: Cloud Master Service Agreement taxonomy

We refer to these collectively as the NIST Cloud Computing Reference Architecture and Taxonomy group output. This same architecture has been used as an underlying organizational pattern for the work of the NIST Cloud Standards Roadmap group in preparing the draft roadmap (NIST SP 500-293 Volume III).

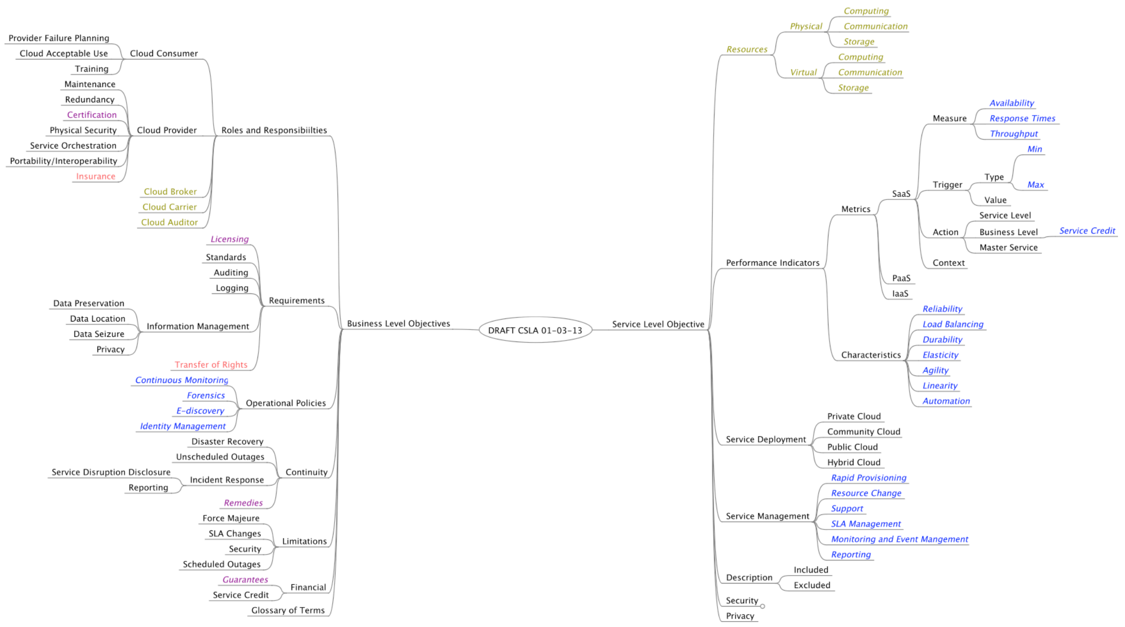


Figure 6: Draft Cloud Service Level Agreement taxonomy

8.2 NIST Cloud Computing Security

The NIST Cloud Computing Security working group has also produced an extensive white paper, and has codified some of the related concepts into a listing of Cloud Computing Impediments and Mediations. Figure 7 gives a mind map derived from this listing in a format suitable for comparison with the SAJACC Use Case and other NIST Cloud Computing working group output discussed above.

8.3 NIST Business Use Cases

Several broadly based US Government use cases were gathered by the NIST Cloud Computing Business Use Cases group, and were referred to by the SAJACC group during preparation of the technical use cases presented in previous sections. A close collaboration between the two groups will also be helpful in future SAJACC activities.

8.4 US VA Bronze, Silver and Gold Use Cases

A valuable analysis of the NIST SAJACC use cases was provided by the US Department of Veterans Affairs, which adopted the general concept and construction of the SAJACC use cases, but extended them to include other sections, including an overall identification and tracking block and sections on “Background”, “Definitions”, “Concept of Operations”, “Primary Actors”, “Business Goal”, “Service Model”, “Deployment Model”, “Necessary Conditions”, “Priorities and Risks”, “Essential Characteristics”, “Normal Flow”, “Frequency of Use”, “Special Requirements”, “Notes and Issues”, and a “Risk Register” comprising a table detailing a description, likelihood, severity, countermeasures and status for each risk identified.

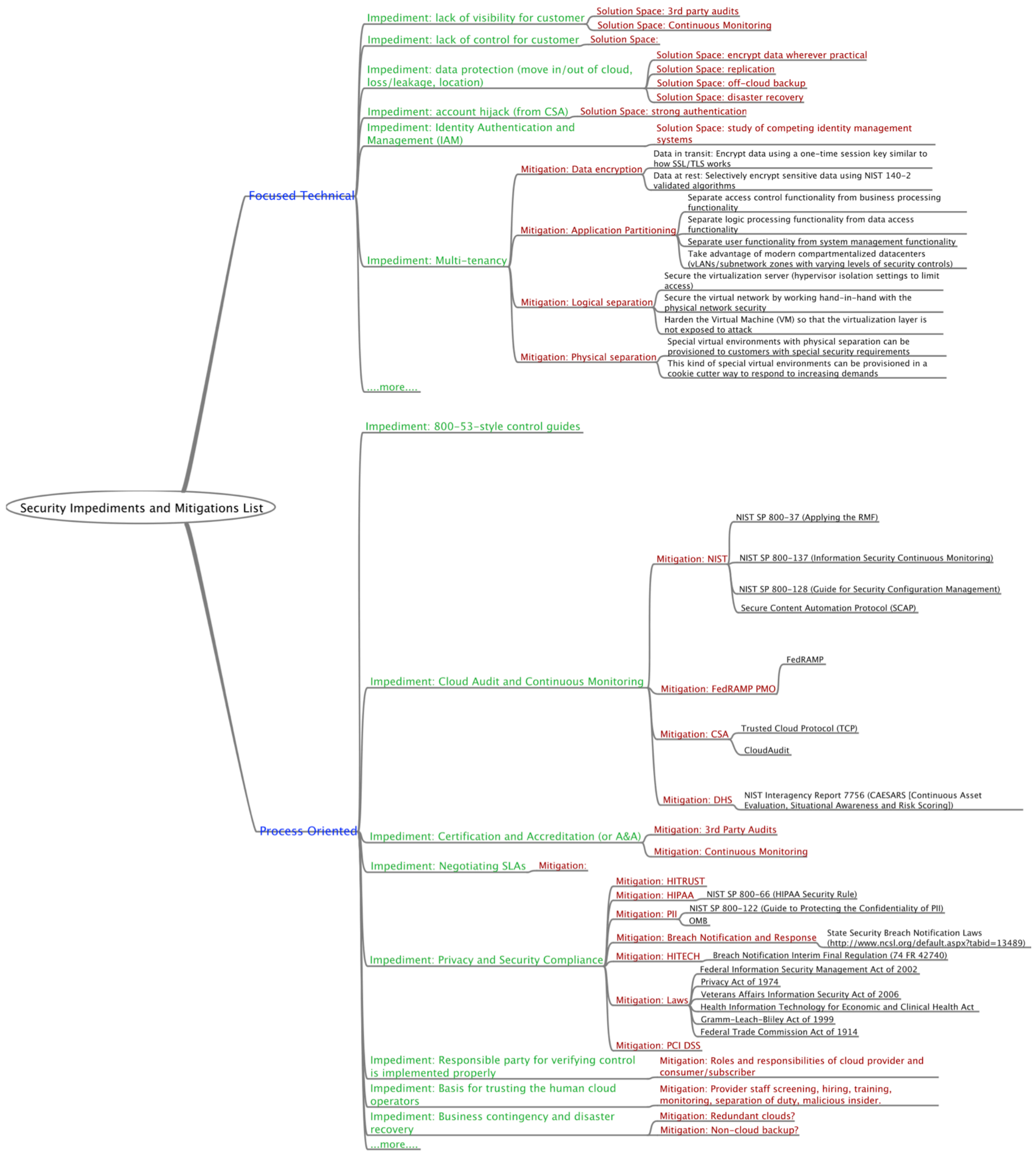

Security Impediments and Mitigations List
Focused Technical
Impediment: lack of visibility for customer
Solution Space: 3rd party audits
Solution Space: Continuous Monitoring
Impediment: lack of control for customer
Solution Space:
Impediment: data protection (move in/out of cloud, loss/leakage, location)
Solution Space: encrypt data wherever practical
Solution Space: replication
Solution Space: off-cloud backup
Solution Space: disaster recovery
Impediment: account hijack (from CSA)
Solution Space: strong authentication
Impediment: Identity Authentication and Management (IAM)
Solution Space: study of competing identity management systems
Impediment: Multi-tenancy
Mitigation: Data encryption
Data in transit: Encrypt data using a one-time session key similar to how SSL/TLS works
Data at rest: Selectively encrypt sensitive data using NIST 140-2 validated algorithms
Mitigation: Application Partitioning
Separate access control functionality from business processing functionality
Separate logic processing functionality from data access functionality
Separate user functionality from system management functionality
Take advantage of modern compartmentalized datacenters (vLANs/subnetwork zones with varying levels of security controls)
Mitigation: Logical separation
Secure the virtualization server (hypervisor isolation settings to limit access)
Secure the virtual network by working hand-in-hand with the physical network security
Harden the Virtual Machine (VM) so that the virtualization layer is not exposed to attack
Mitigation: Physical separation
Special virtual environments with physical separation can be provisioned to customers with special security requirements
This kind of special virtual environments can be provisioned in a cookie cutter way to respond to increasing demands
....more....
Process Oriented
Impediment: 800-53-style control guides
Impediment: Cloud Audit and Continuous Monitoring
Mitigation: NIST
NIST SP 800-37 (Applying the RMF)
NIST SP 800-137 (Information Security Continuous Monitoring)
NIST SP 800-128 (Guide for Security Configuration Management)
Secure Content Automation Protocol (SCAP)
Mitigation: FedRAMP PMO
FedRAMP
Mitigation: CSA
Trusted Cloud Protocol (TCP)
CloudAudit
Mitigation: DHS
NIST Interagency Report 7756 (CAESARS [Continuous Asset Evaluation, Situational Awareness and Risk Scoring])
Impediment: Certification and Accreditation (or A&A)
Mitigation: 3rd Party Audits
Mitigation: Continuous Monitoring
Impediment: Negotiating SLAs
Mitigation:
Impediment: Privacy and Security Compliance
Mitigation: HITRUST
Mitigation: HIPAA
NIST SP 800-66 (HIPAA Security Rule)
Mitigation: PII
NIST SP 800-122 (Guide to Protecting the Confidentiality of PII)
OMB
Mitigation: Breach Notification and Response
State Security Breach Notification Laws (http://www.ncsl.org/default.aspx?tabid=13489)
Mitigation: HITECH
Breach Notification Interim Final Regulation (74 FR 42740)
Mitigation: Laws
Federal Information Security Management Act of 2002
Privacy Act of 1974
Veterans Affairs Information Security Act of 2006
Health Information Technology for Economic and Clinical Health Act
Gramm-Leach-Bliley Act of 1999
Federal Trade Commission Act of 1914
Mitigation: PCI DSS
Impediment: Responsible party for verifying control is implemented properly
Mitigation: Roles and responsibilities of cloud provider and consumer/subscriber
Impediment: Basis for trusting the human cloud operators
Mitigation: Provider staff screening, hiring, training, monitoring, separation of duty, malicious insider.
Impediment: Business contingency and disaster recovery
Mitigation: Redundant clouds?
Mitigation: Non-cloud backup?
...more....

Figure 7: NIST Cloud Computing Security working group Impediments and Mitigations

The US VA use cases were further categorized into Bronze, Silver and Gold level designations depending on these considerations and the degree to which the cloud functionality described in each use case would be hosted within or have access to US VA agency resources.

A mind map showing the US VA use cases categorized into Bronze, Silver and Gold levels is shown in Figure 8.

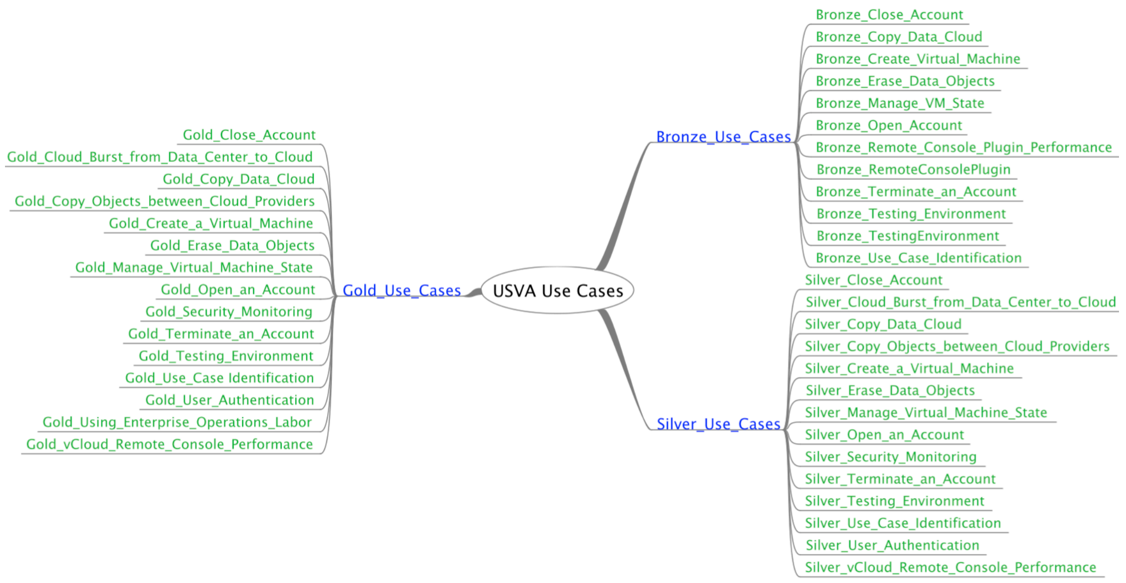

USVA Use Cases
Bronze_Use_Cases
Bronze_Close_Account
Bronze_Copy_Data_Cloud
Bronze_Create_Virtual_Machine
Bronze_Erase_Data_Objects
Bronze_Manage_VM_State
Bronze_Open_Account
Bronze_Remote_Console_Plugin_Performance
Bronze_RemoteConsolePlugin
Bronze_Terminate_an_Account
Bronze_Testing_Environment
Bronze_TestingEnvironment
Bronze_Use_Case_Identification
Silver_Use_Cases
Silver_Close_Account
Silver_Cloud_Burst_from_Data_Center_to_Cloud
Silver_Copy_Data_Cloud
Silver_Copy_Objects_between_Cloud_Providers
Silver_Create_a_Virtual_Machine
Silver_Erase_Data_Objects
Silver_Manage_Virtual_Machine_State
Silver_Open_an_Account
Silver_Security_Monitoring
Silver_Terminate_an_Account
Silver_Testing_Environment
Silver_Use_Case_Identification
Silver_User_Authentication
Silver_vCloud_Remote_Console_Performance
Gold_Use_Cases
Gold_Close_Account
Gold_Cloud_Burst_from_Data_Center_to_Cloud
Gold_Copy_Data_Cloud
Gold_Copy_Objects_between_Cloud_Providers
Gold_Create_a_Virtual_Machine
Gold_Erase_Data_Objects
Gold_Manage_Virtual_Machine_State
Gold_Open_an_Account
Gold_Security_Monitoring
Gold_Terminate_an_Account
Gold_Testing_Environment
Gold_Use_Case Identification
Gold_User_Authentication
Gold_Using_Enterprise_Operations_Labor
Gold_vCloud_Remote_Console_Performance

Figure 8: Bronze, Silver and Gold use cases identified from SAJACC use case examples by the US Department of Veterans Affairs

8.5 Other Sources of Input

In addition to the official NIST Cloud Computing working group output, the SAJACC group recruited and received presentation of reports from other community sectors during its working group meetings. These are recorded in the SAJACC meeting materials pages, and notably included public documents provided by several standards development organizations, including OGF, DMTF and TM Forum, as well as other public documentation provided by working group participants (references [DMTF], [IETF-SCIM], [LIBCLOUD], [NSTIC], [OGF], [ORACLE], [SNIA], [TMFORUM]). Several of these have resulted in candidate use cases that are still under development and that might fit into an expanded SAJACC Use Case set as described below, but are not yet to the maturity level of the other use cases included in this report.

8.6 Analysis and Recommendations for SAJACC Use Case Reorganization

Comparison of the mind map diagrams derived from the above input with the SAJACC use case organization given in Sections 1-2 shows the opportunity for reorganization of the use case scenarios into a form that would be more compatible with the reference architecture and with the taxonomies produced by the other NIST Cloud Computing working groups. There are several points of overlap between Figure 4, for example, and Figure 2. Specifically, the Cloud Services Management, Cloud Broker and Security portions of these diagrams show several points of contact, and components in the Cloud Interoperability portion of the SAJACC use cases can be rearranged into sub-branches of the reference architecture taxonomy shown in Figure 4.

It appears necessary to expand the diagram given in Figure 4 with input from the Cloud Service Agreement and Cloud SLA diagrams from Figures 5 and 6, and to further enhance the combination of topics already present in the Security branches of the SAJACC and Reference Architecture diagrams by inclusion of the more comprehensive listing from the NIST Cloud Computing Security group as given in Figure 7.

Several of the branches identified in the service agreement and SLA diagrams have points of contact with the existing SAJACC use case categories, but refinement of the SAJACC diagram to include SLA concepts in particular seems to be necessary. An overall classification could thus be built for future versions of the SAJACC use case scenarios that would both provide a better fit to the overall NIST reference architecture and taxonomy, and also include a more comprehensive listing in the areas of cloud service agreements, SLAs and cloud security considerations.

A recommendation proceeding from the above is to prepare technical use cases that derive from this expanded taxonomy, but that are formulated in terms that are similar to the previous SAJACC use case scenarios, with actors, goals, success scenarios and failure conditions, failure handling, assumptions, requirements and credit identified as for the previously developed use case scenarios in a form that will encourage the use of the SAJACC process for demonstration and validation of related cloud products and standards in terms of their applicability to these scenarios. Considering the examples given in the US VA use cases, a more comprehensive set of sections can be implemented that includes some or all of the categories identified in the US VA set, and optionally could consider a separation into categories depending on the level of access of the cloud functionality to government agency internal resources.

Input from the community at the NIST Cloud Computing and Big Data Forum and Workshop held January 15-17, 2013 reinforced the above points and appeared to represent a broad consensus that the plans of the working group presented above and summarized below represent a strong plan for moving forward. Additional points that were made at the workshop included the need to provide for measurement of performance of various cloud products and standards when meeting the terms of a given technical use case, as well as the corresponding need to gather further input from additional USG agencies and business use cases.

As a final recommendation in light of the fundamental importance of security to all cloud usage scenarios, we recommend the addition of a "Security Considerations" section into all of the SAJACC technical use cases, which should include a risk register of the type described in the US VA use cases, but also have room for explicit inclusion of cloud security impediments and mitigations as described by the NIST Cloud Computing Security working group. Previous SAJACC use cases should be examined and updated to include these entries as part of the reorganization and reclassification, and all future SAJACC use cases should include such entries on an essential basis, along with each of the other use case factors.

9. Conclusions

This report has presented a summary of the concepts and a listing of the technical use cases prepared by the NIST SAJACC working group during the first two years of its operation. An extensive collection of use cases has been prepared, and several of these have been subjected to real-world tests using cloud computing software, both open source and commercial, and also to comparisons with capabilities provided by several cloud computing standards. These scenarios were then refined through additional interactions with US Federal Government agencies, notably the US Department of Veterans Affairs, and extended to meet their needs for a multi-tiered system divided in terms of levels of support, security and access to USVA resources.

The process has been sufficient to identify a major set of cloud computing capabilities that are amenable to validation testing for the applicability of cloud standards, standards-based products, and individual cloud product APIs to assess the capabilities of these standards and products against these use cases. Several such demonstrations were conducted by the SAJACC group itself during the first portion of this work, and a pattern for conducting validation tests as described in Section 1 of this report has been developed.

To further improve the SAJACC process, we propose that work be done on a continuing basis by the SAJACC group to expand and reorganize the SAJACC use case listings, using the output of other NIST Cloud Computing working groups as a guide. We also recommend expanding the set of use cases to make them more complete with respect to the NIST Cloud Computing Reference Architecture and Taxonomy, Business Use Cases, Security and Standards Roadmap output, and that each of the technical use cases be extended to include a "Security Considerations" section.

In list form, the recommendations of the SAJACC working group proceeding from this phase of its efforts are as follows:

1. Replace the SAJACC use case internal organization with one based on the current structure of the NIST Cloud Computing Reference Architecture and Taxonomy;
2. Add further use cases based on current extensions to this taxonomy for recently developed Cloud SLA Metrics and NIST Cloud Computing Security components;
3. Integrate further input as necessary from the NIST Business Use Case and Standards Roadmap groups, and work closely with these groups to identify additional use cases;
4. Study and adopt use case template elements from the US VA Bronze, Silver and Gold Use Cases and from additional formal input from US Government agencies;
5. Add automation and tooling, if possible, to the NIST web site to support community downloading of the NIST SAJACC use cases and their associated templates for testing scenarios and uploading of externally produced test results;
6. Conduct, invite and document additional use case demonstrations of cloud standards and applicable products against the SAJACC use cases to illustrate their features;
7. Solicit and add further recommendations from the community at large through meetings of the SAJACC working group.

This report can therefore be taken to comprise the conclusion of Phase I of the SAJACC process, and the plan for initiation of Phase II with the goal to implement the above recommendations.

In summary, we recommend that the SAJACC process be enhanced with the above adjustments beyond its initial phase and continued to complete the initial design for validation testing, as given in Section 1 of this report, and to extend this design as needed to support a more comprehensive framework for technical use case definition, validation and testing in a format that is consistent with the ongoing work of the rest of the NIST Cloud Computing working groups.

Appendix A: Acronyms and Abbreviations

Selected acronyms and abbreviations used in this report are defined below.

API	Application Programmer Interface
FISMA	Federal Information Security Management Act
IaaS	Infrastructure as a Service
IT	Information Technology
ITL	Information Technology Laboratory
NIST	National Institute of Standards and Technology
NSTIC	National Strategy for Trusted Identities in Cyberspace
PaaS	Platform as a Service
OMB	Office of Management and Budget
REST	Representational State Transfer
SaaS	Software as a Service
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
SP	Special Publication
UDDI	Universal Description, Discovery and Integration
VM	Virtual Machine
WSDL	Web Services Description Language

Appendix B: Glossary

Authentication Credential. Something that an entity is, has, or knows that allows an entity to prove its identity to a system.

Cloud-subscriber. An authenticated person that accesses a cloud system over a network. A **cloud-subscriber** may possess administrative privileges, such as the ability to manage virtual machines, or the ability to regulate access by users to cloud resources the **cloud-subscriber** controls.

Data Object. A logical container of data, that can be accessed over a network. E.g., a blob. May be an archive, such as specified by the TAR format.

Physical Data Container. A storage device physically suitable for transferring data between **cloud-subscriber**s and clouds; e.g., a hard disk. There has to be a standard format that the Provider supports (e.g., EIDE, IDE, SCSI). The physical data container must be formatted with a standard logical organization, such as FAT32, ufs, etc.

Provider. An organization that offers a network service that satisfies the definition of cloud computing given in Section

SLA. A document explaining expected quality of service and legal guarantees. Contains at least the following data fields:

CloseDelay: the minimum latency, expressed in a common time unit, for a cloud provider to respond to a user's request to close an account.

User. A person or computer that accesses a cloud system over a network. A user may be authenticated but can also be anonymous. A user does not have administrative privileges on a cloud system.

Appendix C: Extended Use Cases and Use Case Templates from the US Department of Veterans Affairs

Background and Design Philosophy

The Austin Information Technology Center (AITC) of the US Department of veterans Affairs initiated efforts in 2012 to document use cases to support a VA cloud on the basis of the Cloud First Mandate from the US Chief Information Officer, which requires that each USG agency implement cloud computing services whenever possible. The goals of this effort were to give the VA maximum capacity utilization, improved IT flexibility, and minimized costs. During the startup period, AITC Cloud services were designated only for test and development. A plan is in place based on lessons learned from this period to bring pre-production and production services online in 2013.

The use cases generated by the VA were broken down into four categories, depending on factors that included the level of access to VA internal networks and other related considerations. An initial set of use cases was generated based on the SAJACC use case scenarios documented in the current report. Once experience is gathered from the initial test, development and pre-production phases of deployment, the VO plans to generate additional use cases.

VA Use Case Categories

30 day Free Demo – A free environment with the ability to create a few Virtual Machines (VMs) and then try the environment before you purchase a Bronze, Silver, or Gold environment. This environment provides the ability to load any Operating System with any application to which users have licensed access. User applications and servers do not have to pass any AITC security scans. User teams can modify the environment how and when they like. The environment will be in an isolated sandbox and cannot access other VA domains.

Bronze – Users at the Bronze level can create a sandbox environment, 100% self-service, create VM templates/master images, limited Internet access, and the VMs can't access other VA domains. This environment provides the ability to load any Operating System with any application you have access to. User applications and servers do not have to pass any AITC security scans. User teams can modify the environment how and when they like. The environment will be in an isolated sandbox and cannot access other VA domains.

Silver - Includes Bronze-level services, Limited VA network access, filtered outgoing internet, pool of IP addresses, can join VA domains, DNS assistance, Use provided VM templates/master images, Continuous Security Monitoring, OS patching. This environment provides the ability to load a hardened VA build of Windows 2008 and RHEL. User VMs are scanned for vulnerabilities by AITC Enterprise Operations. User teams can modify the environment how and when they like. This environment is able to join other VA domains.

Gold - Includes Bronze and Silver level services, Infrastructure monitoring, access to Enterprise Operations labor resources to accelerate timelines. This environment provides the ability to load a hardened VA build of Windows 2008 and RHEL. User VMs are scanned for vulnerabilities by AITC Enterprise Operations. User teams can modify the environment how and when they like.

The environment is able to join other VA domains. The ability to engage Enterprise Operations labor resources to accelerate deliverables and Infrastructure monitoring is included.

Characteristics of VA Use Case Templates

When comparing the use cases prepared by the Department of Veterans Affairs to those of the original SAJACC examples, the SAJACC group noted the addition of several standard fields and an extended range of documentation regarding potential risks and mitigations in the form of a "risk register". Other features of the VA use cases included delineation of the service and deployment models to be used, a section on "concept of operations" and a description of necessary conditions.

The SAJACC group felt that these components of the VA use case template patterns are each useful additions to the format of the basic SAJACC use cases already assembled. As a result, the recommendation of the SAJACC working group is to incorporate similar changes, including addition of a section on "security considerations" that can include the "risk register" and other additional sections, in a future review and update of the basic SAJACC use case scenarios.

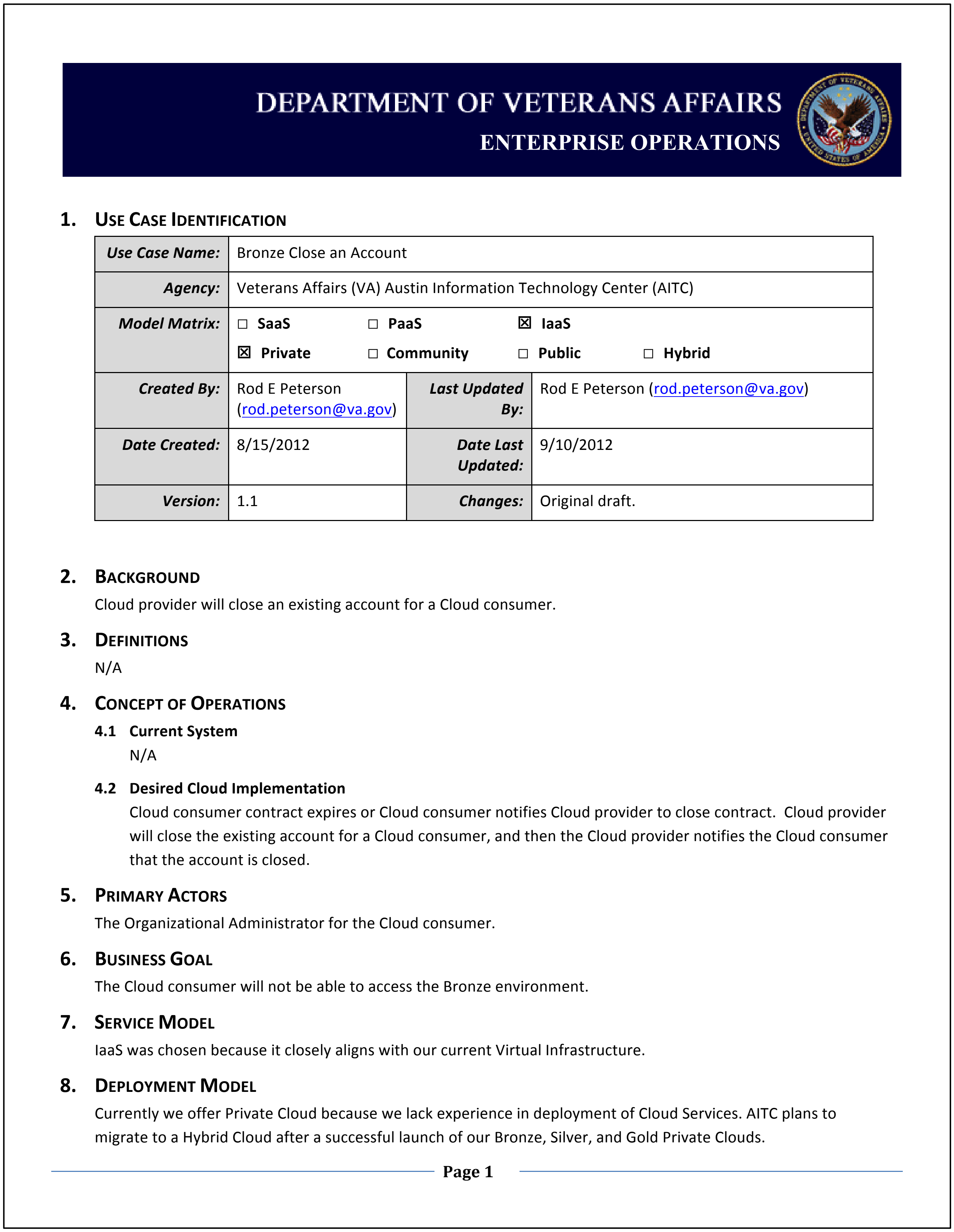
Listing of Current VA Use Cases

The following table summarizes the VA use cases that have been created to date.

Bronze	Silver	Gold
Close Account	Close Account	Close Account
Copy Data To/From Cloud	Cloud Burst from Data Center to Cloud	Cloud Burst from Data Center to Cloud
Create Virtual Machine	Copy Data To/From Cloud	Copy Data To/From Cloud
Erase Data Objects	Copy Objects between Cloud Providers	Copy Objects between Cloud Providers
Manage VM State	Create a Virtual Machine	Create a Virtual Machine
Open Account	Erase Data Objects	Erase Data Objects
Remote Console Plugin Performance	Manage Virtual Machine State	Manage Virtual Machine State
Terminate an Account	Open an Account	Open an Account
Testing Environment	Security Monitoring	Security Monitoring
Use Case Identification	Terminate an Account	Terminate an Account
	Testing Environment	Testing Environment
	Use Case Identification	Use Case Identification
	Remote Console Plugin Performance	User Authentication

Example VA Use Case

The following pages document an example of one of the VA use cases for purposes of illustrating their component features. A full set of these use cases can be downloaded from the SAJACC meeting materials pages for the series of meetings held in late 2012.



DEPARTMENT OF VETERANS AFFAIRS

ENTERPRISE OPERATIONS

1. Use Case Identification

Use Case Name:	Bronze Close an Account		
Agency:	Veterans Affairs (VA) Austin Information Technology Center (AITC)		
Model Matrix:	☐ **SaaS** ☐ **PaaS** ☒ **IaaS** ☒ **Private** ☐ **Community** ☐ **Public** ☐ **Hybrid**		
Created By:	Rod E Peterson (rod.peterson@va.gov)	***Last Updated By:***	Rod E Peterson (rod.peterson@va.gov)
Date Created:	8/15/2012	***Date Last Updated:***	9/10/2012
Version:	1.1	***Changes:***	Original draft.

2. Background

Cloud provider will close an existing account for a Cloud consumer.

3. Definitions

N/A

4. Concept of Operations

4.1 Current System

N/A

4.2 Desired Cloud Implementation

Cloud consumer contract expires or Cloud consumer notifies Cloud provider to close contract. Cloud provider will close the existing account for a Cloud consumer, and then the Cloud provider notifies the Cloud consumer that the account is closed.

5. Primary Actors

The Organizational Administrator for the Cloud consumer.

6. Business Goal

The Cloud consumer will not be able to access the Bronze environment.

7. Service Model

IaaS was chosen because it closely aligns with our current Virtual Infrastructure.

8. Deployment Model

Currently we offer Private Cloud because we lack experience in deployment of Cloud Services. AITC plans to migrate to a Hybrid Cloud after a successful launch of our Bronze, Silver, and Gold Private Clouds.

9. Necessary Conditions

9.1 Security: Cloud consumer must be notified that the account has been closed.

9.2 Interoperability: Cloud consumer will use vCloud Director to access the environment.

9.3 Portability: N/A

9.4 Other: N/A

10. Priorities and Risks

N/A

11. Essential Characteristics

On-demand self-service:	Cloud consumer will have the ability to self-provision resources in Bronze Environment.
Broad network access:	Bronze environment allows network access within Cloud consumer's environment, but does not allow access to other VA networks.
Resource pooling:	Cloud consumer has the ability to create any size VM within their environment total capacity.
Rapid elasticity:	Cloud consumer can expand the resources of any VM within their environment.
Measured service:	Cloud provider does not offer performance/monitoring service within Bronze Environment.

12. Normal Flow

Organizational Administrator will not be able to access the Bronze environment.

13. Frequency of Use

Cloud provider will close the account one time.

14. Special Requirements

N/A

15. Notes and Issues

N/A

16. Risk Register

Date	Description	Likelihood	Severity	Countermeasures	Status
08/15/2012	Cloud provider closes the account too early.	Low (<30%)	High (>70%)	Proper steps defined for Cloud provider to verify an account should be closed.	Current
08/15/2012	Cloud provider overcharges the cloud consumer.	Low (<30%)	Medium (31-70%)	Proper steps defined for Business Office to not charge Cloud consumer after account has been closed.	Current
8/15/2012	Cloud provider fails to notify the Cloud consumer that the account is closed	Low (<30%)	Medium (31-70%)	Proper steps defined for how to notify Cloud consumer that the account is closed.	Current
08/15/2012	Cloud provider fails to revoke the Cloud consumer's authentication information.	Low (<30%)	Medium (31-70%)	Proper steps defined for how to close an account. Cloud provider will need to test that the account is closed and not accessible using proper credentials.	Current

Appendix D: Source Code Listings for Example Use Case Demonstrations

The following pages include source code listings for some of the SAJACC demonstration examples given in Section 7 in the main report above. This is not a complete set of listings, but should be taken only as examples.

The first example given for demonstrations related to use case 3.4 "Copy Data Objects Into A Cloud" includes separate source code for the Amazon S3 and Microsoft Azure products. Other examples are included for virtual machine control using the AWS API and similar Eucalyptus open source equivalent product and for the OGF OCCI standard using an OpenNebula implementation, and for data transfer using a reference implementation of the CDMI standard provided by SNIA.

The source code for the full set of example reports given in Section 7 is not included for the latter two of these examples, since it is largely an aggregation of the code applicable to the separate examples for S3 and CDMI already presented here.

Further downloadable source code files for these and other use case demonstrations are available on the SAJACC meeting materials NIST TWiki pages at http://collaborate.nist.gov/twiki-cloud-computing/bin/view/CloudComputing/SAJACC.

```java
/*
 * DISCLAIMER: Certain commercial entities, equipment, or materials
 * may be identified in this document in order to describe an
 * experimental procedure or concept adequately.  Such identification
 * is not intended to imply recommendation or endorsement by the
 * National Institute of Standards and Technology, nor is it intended
 * to imply that the entities, materials, or equipment are necessarily
 * the best available for the purpose.
 *
 * Design and implementation of this program was paid for by U.S. tax
 * dollars. Therefore it is public domain. However, the author and
 * NIST would appreciate credit if this program or parts of it are
 * used.
 *
 * $Id: UseCase3_4_CDMI.java 65 2011-02-22 17:39:29Z $
 */
package sajacc.usecases;

import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Random;
import java.util.ResourceBundle;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

import com.google.gson.JsonParser;

/**
 * UseCase3_4_CDMI uses [Apache HttpComponents]
 * (http://hc.apache.org/httpcomponents-client-ga/) as a client side
 * HTTP client library to communicate with a CDMI compliant
 * server. The testing code will be a stand-alone Java application and
 * executes the SAJACC use case scenario flows, and report out
 * progress.
 *
 * The report out can use simple text-based markup for better
 * readability. [Markdown](http://en.wikipedia.org/wiki/Markdown)
 * syntax is chosen for its simplicity and readability. Various
 * markdown conversion tools can be used to concert the output into
 * HTML or other format if desired. The report should highlight texts
 * that corresponds to the SAJACC use case text to show mapping of the
 * implementation to the use case.
 *
 */
public class UseCase3_4_CDMI {

    /**
     * The entry point of this test driver.
     *
     */
    public static void main(String[] args) {

        // File name of a test file under data directory to be copied
        // to the cloud.
        String objectName = "helloworld.txt";
        String containerName = "";
        String cdmiServerUrl = "";


        // Generate a random bucket name everytime running the
        // test, to avoid any possible name conflicts.
        Random random = new Random();
        containerName = "test-container-sajacc-usecases-3-4"
            + random.nextInt();

        try {
            report("# NIST Cloud Computing Use Case Testing Report #");
            report("## 3.4 Copy Data Objects Into A Cloud ##");
            report("This test driver implements `Success Scenario 1 " +
                   "(cloud-subscriber-to-network copy, IaaS, PaaS, SaaS)` " +
                   "of [Use Case 3.4](http://www.nist.gov/itl/cloud/3_4.cfm) " +
                   "using CDMI interface.");

            report("Test Scenario:");

            /************************************************************/
            report("* __The cloud-subscriber determines a local file__, `"
                   + objectName
                   + "`, __for copying to the cloud-provider's system__.");
            /************************************************************/

            /************************************************************/
            // Reading from configuration file "cdmi.properties" from
            // classpath, to be used to instantiate a client instance
            ResourceBundle rb = ResourceBundle.getBundle("cdmi");

            // Compose the cdmiServerUrl based on configured
            // properties `cdmi-server-host` and `cdmi-server-port`.
            // In this case, assuming `http` is used and `cdmi-server`
            // is used as the context in the URL.
            cdmiServerUrl = "http://" + rb.getString("cdmi-server-host")
                + ":" + rb.getString("cdmi-server-port") + "/cdmi-server";

            // create a container
            createContainer(cdmiServerUrl, containerName);
            report("* __The cloud-subscriber issues a command to "
                   + "create a container__: `" + containerName + "`.");
            /************************************************************/

            /************************************************************/
            // Read the test object file from "data" directory
            BufferedReader in = new BufferedReader(new FileReader("data/" +
                                                   objectName));

            // reading the content of the text file into a string
            String str;
            String content  = "";
            while ((str = in.readLine()) != null) {
                content += str;
            }

            // create the object in the container
            createTextObject(cdmiServerUrl,
                             containerName,
                             objectName,
                             content);
            report("* __The cloud-subscriber issues a command to "
                   + "create an object in the container created, and "
                   + "transfers the local file__ `" + objectName + "`"
                   + "__to the cloud-provider's system__.");
            /************************************************************/

            /************************************************************/
            report("Verifying test result:");

            HttpResponse response = getObject(cdmiServerUrl,
                                              containerName,
                                              objectName);

            report("* Download Test Object. ");
```

2

```
            // Read downloaded data as a string to show
            String textData = extractEntityElementAsString(response, "value");

            report("* Download Test Object converted to string as: " + textData
                   + ".");

            // Verify the data downloaded against the source file,
            // using the equals method for simplicity.
            boolean valid = textData.equals(content);
            report("* Compare against source file `" + objectName
                   + "` returned `" + valid + "`" + ".");
            /************************************************************/
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                report("Cleaning up testing objects: `" + objectName + "`.");
                deleteObject(cdmiServerUrl, containerName, objectName);
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                report("Cleaning up testing container: `" + containerName
                       + "`.");
                deleteContainer(cdmiServerUrl, containerName);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void report(String message) {
        System.out.println(message + "\n");
    }

    // --------------------------------------------------------
    // Helper methods to talk to a CDMI server over its RESTful
    // interface per CDMI spec.
    // --------------------------------------------------------

    /* create container */
    private static HttpResponse createContainer(String cdmiServerUrl,
                                                String containerName)
        throws Exception {

        HttpClient httpClient = new DefaultHttpClient();

        HttpPut httpput = new HttpPut(cdmiServerUrl + "/" + containerName);
        httpput.setHeader("Content-Type",
                          "application/vnd.org.snia.cdmi.container+json");
        httpput.setHeader("X-CDMI-Specification-Version", "1.0");

        String reqStr = "{\n";
        reqStr += "\"metadata\" : {\n";
        reqStr += "}\n";
        reqStr += "}\n";
        StringEntity inputEentity = new StringEntity(reqStr);
        httpput.setEntity(inputEentity);

        return httpClient.execute(httpput);
    }

    /* create text data object */
    private static HttpResponse createTextObject(String cdmiServerUrl,
                                                 String containerName,
                                                 String objectName,
                                                 String objectContent)
        throws Exception {

        HttpClient httpClient = new DefaultHttpClient();
```

```java
        HttpPut httpput = new HttpPut(cdmiServerUrl + "/" + containerName + "/" + objectName);
        httpput.setHeader("Content-Type",
                          "application/vnd.org.snia.cdmi.dataobject+json");
        httpput.setHeader("X-CDMI-Specification-Version", "1.0");
        String reqStr = "{\n";
        reqStr += "\"mimetype\" : \"" + "text/plain" + "\",\n";
        reqStr += "\"value\" : \"" + objectContent + "\"\n";
        reqStr += "}\n";
        StringEntity entity = new StringEntity(reqStr);
        httpput.setEntity(entity);
        return httpClient.execute(httpput);
    }

    /* get data object */
    private static HttpResponse getObject(String cdmiServerUrl,
                                          String containerName,
                                          String objectName)
        throws Exception {

        HttpClient httpClient = new DefaultHttpClient();
        HttpGet httpget = new HttpGet(cdmiServerUrl + "/" +
                                      containerName + "/" +
                                      objectName);

        httpget.setHeader("Accept",
                          "application/vnd.org.snia.cdmi.dataobject+json");
        httpget.setHeader("Content-Type",
                          "application/vnd.org.snia.cdmi.dataobject+json");
        httpget.setHeader("X-CDMI-Specification-Version", "1.0");
        return httpClient.execute(httpget);
    }

    /* delete data object */
    private static HttpResponse deleteObject(String cdmiServerUrl,
                                     String containerName,
                                     String objectName)
        throws Exception {

        HttpClient httpClient = new DefaultHttpClient();
        HttpDelete httpdelete = new HttpDelete(cdmiServerUrl + "/" +
                                         containerName + "/" +
                                         objectName);
        httpdelete.setHeader("Content-Type",
                             "application/vnd.org.snia.cdmi.dataobject+json");
        httpdelete.setHeader("X-CDMI-Specification-Version", "1.0");
        return httpClient.execute(httpdelete);
    }

    /* delete container */
    private static HttpResponse deleteContainer(String cdmiServerUrl,
                                                String containerName)
        throws Exception {

        HttpClient httpClient = new DefaultHttpClient();
        HttpDelete httpdelete = new HttpDelete(cdmiServerUrl + "/" +
                                         containerName);
        httpdelete.setHeader("Content-Type",
                             "application/vnd.org.snia.cdmi.dataobject+json");
        httpdelete.setHeader("X-CDMI-Specification-Version", "1.0");
        return httpClient.execute(httpdelete);
    }

    // ---------------------------------------------------------
    // Simple utility method to extract CDMI entity payload JSON
    // element using a Json parser to convenience.
    // ---------------------------------------------------------
    private static String extractEntityElementAsString(HttpResponse response,
                                                       String elementName)
        throws Exception {
        JsonParser parser = new JsonParser();
        String payloadJson = EntityUtils.toString(response.getEntity());
```

```
            return parser
                .parse(payloadJson)
                .getAsJsonObject()
                .get(elementName)
                .getAsString();
        }

    }

```

```
/**
 * Copyright  2006-2010 Soyatec
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * $Id$
 */
package azure.usecases;

import java.io.FileInputStream;
import java.net.URI;
import java.util.Random;
import java.util.ResourceBundle;

import org.jets3t.service.utils.ServiceUtils;
import org.soyatec.windowsazure.blob.BlobStorageClient;
import org.soyatec.windowsazure.blob.IBlobContainer;
import org.soyatec.windowsazure.blob.IBlobContents;
import org.soyatec.windowsazure.blob.IBlobProperties;
import org.soyatec.windowsazure.blob.IBlockBlob;
import org.soyatec.windowsazure.blob.internal.BlobContents;
import org.soyatec.windowsazure.blob.internal.BlobProperties;
import org.soyatec.windowsazure.blob.io.BlobFileStream;
import org.soyatec.windowsazure.blob.io.BlobMemoryStream;
import org.soyatec.windowsazure.internal.util.TimeSpan;

/**
 * UseCase3_4 uses [windowsazure4j] (http://www.windowsazure4j.org/) Java client
 * side library to communicate with a Windows Azure compliant implementation.
 * The implementation is meant for feasibility demonstration.
 *
 * The testing code will be a stand-alone Java application and executes the
 * SAJACC use case scenario flows, and report out progress.
 *
 * The report out can use simple text-based markup for better readability.
 * [Markdown](http://en.wikipedia.org/wiki/Markdown) syntax is chosen for its
 * simplicity and readability. Various markdown conversion tools can be used to
 * concert the output into HTML or other format if desired. The report should
 * highlight texts that corresponds to the SAJACC use case text to show mapping
 * of the implementation to the use case.
 *
 */
public class UseCase3_4_Azure {

    /**
     * The entry point of this test driver.
     *
     */
    public static void main(String[] args) {
        // File name of a test file under data directory to be copied
        // to the cloud.
        String objectName = "helloworld.txt";

        // Generate a random container name everytime running the test,
        // to avoid any possible name conflicts.
        Random random = new Random();
        String containerName = "test-container-azure-usecases-3-4"
                + random.nextInt();

        IBlobContainer container = null;
        IBlockBlob blob = null;

        // Handle to windows azure storage
        BlobStorageClient storage = null;

        report("# Windowsazure4j Cloud Computing Use Case Testing Report #");
        report("## 3.4 Copy Blob Data Into A Cloud ##");
        report("Test Scenario - `Success Scenario 1`:");

        /********************************************************/
        report("* __The user determines a local file__, `" + objectName
                + "`, __for copying to the windows azure storage.");
        /********************************************************/

        try {
            /********************************************************/
            // Reading configuration file "azure.properties"
            // from classpath, to be used to instantiate a Windows
            // Azure Storage execution context
            ResourceBundle rb = ResourceBundle.getBundle("azure");
            storage = BlobStorageClient.create(
                    URI.create("http://blob.core.windows.net"), false,
                    rb.getString("storage-name"), rb.getString("storage-key"));

            report("* The user creates a Windows Azure Storage execution context in "
                    + "preparation to issue commands to the windows azure storage "
                    + "using pre-acquired credentials.");
            /********************************************************/

            /********************************************************/
            // Create a container
            container = storage.createContainer(containerName);
            report("* __The user issues a command to "
                    + "create a container__: `" + containerName + "`.");
            /*********************************************************/

            /*********************************************************/
            // Create the Azure blob on the windows auzre storage and transfer
            // the local file object
            IBlobProperties blobProperties = new BlobProperties(objectName);
            String contentMD5 = ServiceUtils.toBase64(ServiceUtils
                    .computeMD5Hash(new FileInputStream("data/"
                            + objectName)));
            blobProperties
                    .setContentMD5(contentMD5);

            IBlobContents blobContents = new BlobContents(new BlobFileStream("data/"
                    + objectName));
            blob = container.createBlockBlob(blobProperties, blobContents);

            report("* __The user issues a command to "
                    + "create a blob in the container created, and "
                    + "transfers the local file__ `" + objectName + "`"
                    + "__to this blob__.");
            /*********************************************************/

            /*********************************************************/
            report("Verifying test result:");

            // Test verification: retrieve the blob created earlier
            IBlockBlob downloadedBlob = storage.getBlobContainer(containerName)
                    .getBlockBlobReference(objectName);


            report("* Download Test Blob: " + downloadedBlob + ".");

            // Read downloaded data as a string
            BlobMemoryStream stream = new BlobMemoryStream();
            container.setTimeout(TimeSpan.fromSeconds(300));
            downloadedBlob.getContents(stream);
            String textData = new String(stream.getBytes());

            report("* Download Test Blob converted to string as: " + textData
                    + ".");

            // Verify the data downloaded against the source file
            // performing MD5 hash comparison
            String remote = downloadedBlob.getProperties().getContentMD5();
            String local = contentMD5;
            boolean valid = remote.equals(local);

            report("* MD5 hash comparison against source file `" + objectName
                    + "` returned `" + valid + "`" + ".");
            /*********************************************************/
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            report("Cleaning up testing blob: `" + objectName + "`.");
            container.deleteBlob(objectName);

            report("Cleaning up testing container: `" + containerName + "`.");
            storage.deleteContainer(containerName);
        }

    }

    public static void report(String message) {
        System.out.println(message + "\n");
    }
}
```

```
/*
 * DISCLAIMER: Certain commercial entities, equipment, or materials
 * may be identified in this document in order to describe an
 * experimental procedure or concept adequately.  Such identification
 * is not intended to imply recommendation or endorsement by the
 * National Institute of Standards and Technology, nor is it intended
 * to imply that the entities, materials, or equipment are necessarily
 * the best available for the purpose.
 *
 * Design and implementation of this program was paid for by U.S. tax
 * dollars. Therefore it is public domain. However, the author and
 * NIST would appreciate credit if this program or parts of it are
 * used.
 *
 * $Id: UseCase3_6_S3.java 70 2011-03-01 14:54:46Z tongji $
 */
package sajacc.usecases;

import java.io.File;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2Client;
import com.amazonaws.services.ec2.model.DescribeInstancesRequest;
import com.amazonaws.services.ec2.model.DescribeInstancesResult;
import com.amazonaws.services.ec2.model.Instance;
import com.amazonaws.services.ec2.model.Reservation;
import com.amazonaws.services.ec2.model.RunInstancesRequest;
import com.amazonaws.services.ec2.model.RunInstancesResult;
import com.amazonaws.services.ec2.model.TerminateInstancesRequest;
import com.jcraft.jsch.Channel;
import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.Session;
import com.jcraft.jsch.UserInfo;

/**
 * UseCase3_7 uses [AWS SDK for Java]
 * (http://aws.amazon.com/sdkforjava/)
 * to communicate with an EC2 compliant implementation.
 * The implementation is meant for feasibility demonstration.
 *
 * The testing code will be a stand-alone Java application and
 * executes the SAJACC use case scenario flows, and report out
 * progress.
 *
 * The report out can use simple text-based markup for better
 * readability.  [Markdown](http://en.wikipedia.org/wiki/Markdown)
 * syntax is chosen for its simplicity and readability. Various
 * markdown conversion tools can be used to concert the output into
 * HTML or other format if desired. The report should highlight texts
 * that corresponds to the SAJACC use case text to show mapping of the
 * implementation to the use case.
 *
 */
public class UseCase3_7_EC2 {

    /**
     * The entry point of this test driver.
     *
     */
    public static void main(String[] args) {

        String instanceId = null;
        Instance instance = null;

```

1

```java
        // Handle to an EC2 implementation
        AmazonEC2 ec2 = null;
        try {
            report("# NIST Cloud Computing Use Case Testing Report #");
            report("## 3.7 VM Control: Allocate VM Instance ##");
            report("This test driver implements `Success Scenario 1 " +
                   "((AllocateVM, IaaS))`" +
                   "of [Use Case 3.7](http://www.nist.gov/itl/cloud/3_7.cfm) "+
                   "using AWS SDK for Java.");

            report("Test Scenario - `Success Scenario`:");

            /************************************************************/
            // Reading from configuration file "ec2.properties"
            // from classpath, to be used to instantiate a client
            // instance with the credentials provided.
            ResourceBundle rb = ResourceBundle.getBundle ("ec2");

            // Instantiate an instance of the account credentials
            // using the access-key and secret-key configured in
            // aws.properties file.
            AWSCredentials credentials = new
                BasicAWSCredentials(rb.getString("accessKey"),
                                    rb.getString("secretKey"));
            ec2 = new AmazonEC2Client(credentials);
            ec2.setEndpoint(rb.getString("public-url"));

            report("* The cloud-subscriber creates a client handle in "+
                   "preparation to issue commands to the cloud-provider's "+
                   "system using pre-acquired credentials.");
            /************************************************************/

            // get ready to create an instance of a VM with some
            // customizations:
            // - using a pre-created public key for root user
            // - configure to use a pre-created security policy to allow
            //   ssh access
            // - specify the VM instance type (virtual hardware profile)
            String keyName = rb.getString("key-name");
            RunInstancesRequest runRequest =
                new RunInstancesRequest(rb.getString("image-id"), 1, 1);
            runRequest.withSecurityGroups("ssh").setKeyName(keyName);
            runRequest.setInstanceType(rb.getString("instance-type"));

            /************************************************************/
            report("* __(1) The cloud-subscriber requests a specific " +
                   "pre-defined Virtual Machine image supplied by the " +
                   "cloud-provider (O/S, CPU cores, memory, and security) " +
                   "and launches new VM instances__.");
            RunInstancesResult result = ec2.runInstances(runRequest);

            List<Instance> instances = result.getReservation().getInstances();
            for ( Instance ainstance: instances ){
                report("    * Started Instance " + ainstance);
                instance = ainstance;
                instanceId = ainstance.getInstanceId();
                break;
            }

            report("    * Waiting for the VM instance: " + instanceId +
                   " to be ready");
            instance = waitForInstanceState(ec2, instanceId, "running");
            report("    * The VM instance is ready, details: " + instance);

            // Wait for 1 minute to give the sshd service some time to start
            // up properly. Depending on the processing power, sshd sometimes
            // takes a while to start.
            Thread.sleep(60000);

            // Issuing command through SSH
            report("* __(3) The cloud-subscriber has secure " +
```

2

```
                    "launching and administration of their VM " +
                    "instance__.");
            report("    * The cloud-subscriber issues shell command " +
                    "`uname -a; uptime` through SSH session " +
                    "extablished to the VM instance using the " +
                    "private key from key pair:." + keyName);

            report("        * Shell command returns: `" +
                    runSsh(instance.getPublicDnsName(),
                            rb.getString("default-user-id"),
                            rb.getString("ssh-user-private-key-file"),
                            "uname -a; uptime" ) +
                    "`");

        } catch (Exception anyOtherEx) {
            anyOtherEx.printStackTrace();
        } finally {
            try {
                if ( instanceId != null )
                 {
                     report("Cleaning up: Terminate the running VM "+
                            "instance: `" + instanceId + "`.");
                     TerminateInstancesRequest terminateRequest =
                         new TerminateInstancesRequest().
                         withInstanceIds(instanceId);
                     ec2.terminateInstances(terminateRequest);
                     waitForInstanceState(ec2, instanceId, "terminated");
                     report("* Instance terminated.");
                 }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void report(String message) {
        System.out.println(message + "\n");
    }

    // Helper method to check the state of the vm and return when the
    // VM state is at wanted state
    private static Instance waitForInstanceState(AmazonEC2 ec2,
                                                  String instanceId,
                                                  String state) {

        DescribeInstancesRequest request =
            new DescribeInstancesRequest().withInstanceIds(instanceId);

        // Checking state and sleep 1 second if state is not desired state.
        // Return null after waiting for 300x1 seconds.
        for ( int i = 0 ; i < 300 ; i++ ) {
            DescribeInstancesResult describeInstancesRequest =
                ec2.describeInstances(request);

            List<Reservation> reservations =
                describeInstancesRequest.getReservations();

            List<Instance> instances = new ArrayList<Instance>();

            for (Reservation reservation : reservations) {
                instances.addAll(reservation.getInstances());
            }
            if ( instances.size() == 1 ) {
                if (instances.get(0).getState().getName().equals(state)) {
                    return instances.get(0);
                }
            }
            else {
                break;
            }
            try {
```

3

```
                Thread.sleep(5000L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    private static String runSsh(String host,
                                 String user,
                                 String privateKeyFileName,
                                 String command) throws Exception {

        report("    * Issue command:" + command + " to " + user + "@" + host);
        JSch jsch = new JSch();

        File file  = new File("config/" + privateKeyFileName);

        jsch.addIdentity(file.getAbsolutePath());

        Session session = jsch.getSession(user, host, 22);
        session.setUserInfo(new MyUserInfo());

        // To bypass public key verification of the newly created
        // host. In practice, the public key from the sshd on the
        // newly launched VM should be obtained through a secure
        // channel.
        java.util.Properties config = new java.util.Properties();
        config.put("StrictHostKeyChecking", "no");
        session.setConfig(config);

        session.connect();
        Channel channel = session.openChannel("exec");

        ((ChannelExec) channel).setCommand(command);
        channel.setInputStream(null);

        InputStream in = channel.getInputStream();

        ((ChannelExec)channel).setErrStream(System.err);
        channel.connect();

        String retString = new String();
        byte[] tmp=new byte[1024];

        while(true){
            while(in.available()>0){
                int i=in.read(tmp, 0, 1024);
                if(i<0)break;
                retString += new String(tmp, 0, i);
            }
            if(channel.isClosed()){
                break;
            }
            try{Thread.sleep(1000);}catch(Exception ee){}
        }

        channel.disconnect();
        session.disconnect();

        return retString;
    }

    public static class MyUserInfo implements UserInfo {

        public MyUserInfo() {
            super();
        }

        @Override
            public String getPassphrase() {
```

4

```
            return null;
        }

        @Override
            public String getPassword() {
            return  null;
        }

        @Override
            public boolean promptPassphrase(String arg0) {
            return false;
        }

        @Override
            public boolean promptPassword(String arg0) {
            return false;
        }

        @Override
            public boolean promptYesNo(String arg0) {
            return true;
        }

        @Override
            public void showMessage(String arg0) {
        }
    }
}
```

```java
/*
 * DISCLAIMER: Certain commercial entities, equipment, or materials
 * may be identified in this document in order to describe an
 * experimental procedure or concept adequately.  Such identification
 * is not intended to imply recommendation or endorsement by the
 * National Institute of Standards and Technology, nor is it intended
 * to imply that the entities, materials, or equipment are necessarily
 * the best available for the purpose.
 *
 * Design and implementation of this program was paid for by U.S. tax
 * dollars. Therefore it is public domain. However, the author and
 * NIST would appreciate credit if this program or parts of it are
 * used.
 *
 */
package sajacc.usecases;

import java.io.File;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.protocol.ClientContext;
import org.apache.http.HttpHost;
import org.apache.http.util.EntityUtils;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.auth.BasicScheme;
import org.apache.http.impl.client.BasicAuthCache;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicHeader;
import org.apache.http.protocol.BasicHttpContext;
import org.apache.http.protocol.HTTP;

import com.jcraft.jsch.Channel;
import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.Session;
import com.jcraft.jsch.UserInfo;

/**
 * UseCase3_7_OCCI uses [Apache HttpComponents]
 * (http://hc.apache.org/httpcomponents-client-ga/) as a client side
 * HTTP client library to communicate with an OCCI interface
 * endpoint. This test driver is written and tested against an OCCI
 * interface endpoint exposed through an OpenNebula (v2.2)
 * installation. Some OCCI communication messages are OpenNebula
 * environment specific.
 *
 * The testing code will be a stand-alone Java application and
 * executes the SAJACC use case scenario flows, and reports out
 * progress.
 *
 * The report out uses simple text-based markup for better
 * readability.  [Markdown](http://en.wikipedia.org/wiki/Markdown)
 * syntax is chosen for its simplicity and readability. Various
 * markdown conversion tools can be used to concert the output into
 * HTML or other format if desired. The report should highlight texts
 * that corresponds to the SAJACC use case text to show mapping of the
 * implementation to the use case.
 *
 * @author Knowcean Consulting, Prepared for NIST SAJACC Project
 * @author NIST
 */
```

1

```
public class UseCase3_7_OCCI {

    /**
     * The entry point of this test driver.
     *
     */
    public static void main(String[] args) {

        String instanceId = null;

        report("# NIST Cloud Computing Use Case Testing Report #");
        report("## 3.7 VM Control: Allocate VM Instance ##");
        report("This test driver implements `Success Scenario 1 " +
               "((AllocateVM, IaaS))`" +
               "of [Use Case 3.7](http://www.nist.gov/itl/cloud/3_7.cfm) "+
               "using OCCI API.");

        report("Test Scenario - `Success Scenario`:");

        /************************************************************/
        // Reading from configuration file "occi.properties"
        // from classpath, to be used to instantiate client
        // requests with the credentials provided.
        ResourceBundle rb = ResourceBundle.getBundle ("occi");

        // Retrieve all relevant environment specific properties
        // configured.

        // Credentials
        String occiUserName = rb.getString("occi-username");
        String occiUserPassword = rb.getString("occi-password");

        // occi endpoint URL
        String occiPublicUrl = rb.getString("public-url");

        // ====================================
        // compute related sources
        // ====================================

        // instance type of the compute resource
        String instanceType = rb.getString("instance-type");

        // network identifier for network resources that are
        // previously provisioned.
        String networkResourceId = rb.getString("network-id");

        // storage identifier for storage resources that are
        // previously provisioned. In the case of OpenNebula, this is
        // the storage resource ("image") that is a bootable OS image.
        String storageResourceId = rb.getString("storage-id");

        // ====================================
        // target compute resource requirements
        // ====================================

        // type of compute resource
        String computeResourceNumberOfCPU =
            rb.getString("compute-number-of-cpu");
        String computeResourceMemory =
            rb.getString("compute-memory");

        // OpenNebula supports the request of assigning a public IP to
        // a compute resource, assuming it is already under the
        // control on the server side.
        String computeResourcePublicIP = rb.getString("public-ip");

        // ====================================
        // customization properties
        // ====================================

        // OpenNebua supports customizing the VM instance using
```

2

```
        // contextualization data, including some initial scripts
        // to run and user public key for ssh access, which are
        // stored on the cloud server already. This section might
        // be OpenNebula specific and can be passed through OCCI
        // interface.
        //
        String customizationDataPath =
            rb.getString("contextualization-data");
        String vmUserName =
            rb.getString("vm-user-name");
        String vmUserPublicKey =
            rb.getString("vm-user-public-key-file");
        try {
            /***********************************************************/
            report("* __(1) The cloud-subscriber requests a specific " +
                   "pre-defined Virtual Machine image supplied by the " +
                   "cloud-provider (O/S, CPU cores, memory, and security) " +
                   "and launches new VM instances__.");

            // ====================================
            // compose a compute resource creation request
            // ====================================
            String occiComputeResourceCreationRequest =
                "<COMPUTE>" +
                "   <NAME>sajacc-test-vm</NAME>" +
                "   <INSTANCE_TYPE>" + instanceType + "</INSTANCE_TYPE>" +
                "   <DISK>" +
                "     <STORAGE href=\"" +
                occiPublicUrl + "/storage/" + storageResourceId + "\"/>" +
                "   </DISK>" +
                "   <NIC>" +
                "     <NETWORK href=\"" +
                occiPublicUrl + "/network/" + networkResourceId + "\"/>" +
                "   </NIC>" +
                "   <CONTEXT>" +
                "     <HOSTNAME>sajacc-test-vm</HOSTNAME>" +
                "     <IP_PUBLIC>" + computeResourcePublicIP + "</IP_PUBLIC>" +
                "     <FILES>" + customizationDataPath + "</FILES>" +
                "     <TARGET>hdc</TARGET>" +
                //"     <ROOT_PUBKEY>" + vmUserPublicKey + "</ROOT_PUBKEY>" +
                "     <USERNAME>" + vmUserName + "</USERNAME>" +
                "     <USER_PUBKEY>" + vmUserPublicKey + "</USER_PUBKEY>" +
                "   </CONTEXT>" +
                "</COMPUTE>";

            String cloudResponse =
                httpPostToCloud(occiPublicUrl + "/compute",
                                occiUserName,
                                occiUserPassword,
                                occiComputeResourceCreationRequest);

            instanceId = regexExtract(".*<ID>(.*)</ID>.*", cloudResponse);
            if ("".equals(instanceId)) {
                throw new Exception("Failed to create vm: " + cloudResponse);
            }
            report("    * Started Instance `" + instanceId + "`");

            String instance =
                waitForInstanceState(occiPublicUrl + "/compute/" + instanceId,
                                     occiUserName,
                                     occiUserPassword,
                                     "ACTIVE");

            report("    * The VM instance is ready, details: `" +
                   instance + "`");

            // Wait for 1 minute to give the sshd service some time to
            // start up properly. Depending on the processing power,
            // sshd sometimes takes a while to start.
            Thread.sleep(60000);
```

3

```
            // Issuing command through SSH
            report("* __(3) The cloud-subscriber has secure " +
                   "launching and administration of their VM " +
                   "instance__.");
            report("    * The cloud-subscriber issues shell command " +
                   "`uname -a; uptime` through SSH session " +
                   "established to the VM instance using the " +
                   "private key from key pair.");

            report("        * Shell command returns: `" +
                   runSsh(computeResourcePublicIP,
                          vmUserName,
                          rb.getString("vm-user-private-key-file"),
                          "uname -a; uptime" ) +
                   "`");

        } catch (Exception anyOtherEx) {
            anyOtherEx.printStackTrace();
        } finally {
            try {
                if ( instanceId != null )
                {
                    report("Cleaning up: Terminate the running VM "+
                           "instance: `" + instanceId + "`.");
                    httpDeleteFromCloud(occiPublicUrl +
                                        "/compute/" +
                                        instanceId,
                                        occiUserName,
                                        occiUserPassword);
                    report("* Instance terminated.");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    public static void report(String message) {
        System.out.println(message + "\n");
    }

    // Helper method to extract a sub-string from a piece of text
    // using a regular expression.
    private static String regexExtract(String regex, String source) {
        Pattern p = Pattern.compile(regex);
        Matcher m = p.matcher(source);
        if (m.matches()) {
            return m.group(1);
        }
        else {
            return "";
        }
    }

    // Helper method to check the state of the vm and return when the
    // VM state is at wanted state. It polls the httpGetUrl endpoint
    // by issuing HTTP GET requests and sleeps a bit between each
    // poll. Bails out and throws an Exception is waited for too long.
    private static String waitForInstanceState(String httpGetUrl,
                                               String httpUsername,
                                               String httpUserPassword,
                                               String state)
        throws Exception
    {

        // Checking state and sleep 6 seconds if state is not desired state.
        // Return null after waiting for 20x6 seconds.
        for ( int i = 0 ; i < 20 ; i++ ) {
            String httpGetResponse =
                httpGetFromCloud(httpGetUrl,
                                 httpUsername,
                                 httpUserPassword);
            String currentState = regexExtract(".*<STATE>(.*)</STATE>.*",
                                               httpGetResponse);

            if (state.equalsIgnoreCase(currentState)) {
                return httpGetResponse;
            }

            try {
                Thread.sleep(6000L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        throw new Exception("Timed out waiting for instance state " + state);
    }

    // Helper method to create an HTTP client object.
    private static DefaultHttpClient getHttpClient(String username,
                                                   String password)
        throws Exception {
        DefaultHttpClient client = new DefaultHttpClient();
        client.getCredentialsProvider().
            setCredentials(new AuthScope(AuthScope.ANY_HOST, AuthScope.ANY_PORT),
                           new UsernamePasswordCredentials(username, password));
        return client;
    }

    // HTTP context object to facilitate establishing an HTTP
    // authentication context.
    private static BasicHttpContext localContext = new BasicHttpContext();
    private static BasicHttpContext getHttpContext() {
        return localContext;
    }

    // Helper method to parse out the endpoint URL to create an
    // HttpHost object in HTTP authentication communication.
    private static HttpHost getTargetHost(String url) {
        String host = regexExtract("^http[s]?://([^/:]*)[:]?([0-9]+)?.*", url);
        String portStr = regexExtract("^http[s]?://[^/:]*[:]?([0-9]+)?.*", url);

        HttpHost retVal = null;
        if ("".equals(portStr)) {
            retVal = new HttpHost(host);
        }
        else {
            retVal = new HttpHost(host, Integer.parseInt(portStr));
        }

        // update context to force basic auth for target
        BasicAuthCache authCache = new BasicAuthCache();
        BasicScheme basicAuth = new BasicScheme();
        authCache.put(retVal, basicAuth);
        getHttpContext().setAttribute(ClientContext.AUTH_CACHE, authCache);

        return retVal;
    }

    // Helper method to do an HTTP POST to the URL endpoint and use
    // HTTP authentication.
    private static String httpPostToCloud(String url,
                                          String username,
                                          String password,
                                          String entityStr) throws Exception {

        DefaultHttpClient client = getHttpClient(username, password);
        HttpPost httpPost = new HttpPost(url);
        StringEntity entity = new StringEntity(entityStr);
        entity.setContentType(new BasicHeader(HTTP.CONTENT_TYPE, "text/xml"));
        httpPost.setEntity(entity);
        return EntityUtils.toString(client.execute(getTargetHost(url),
                                                       httpPost,
                                                       getHttpContext()
                                                       ).getEntity());
    }

    // Helper method to do an HTTP GET to the URL endpoint and use
    // HTTP authentication.
    private static String httpGetFromCloud(String url,
                                           String username,
                                           String password)
        throws Exception {
        DefaultHttpClient client = getHttpClient(username, password);
        HttpGet httpGet = new HttpGet(url);
        return EntityUtils.toString(client.execute(getTargetHost(url),
                                                   httpGet,
                                                   getHttpContext()
                                                   ).getEntity());
    }

    // Helper method to do an HTTP DELETE to the URL endpoint and use
    // HTTP authentication.
    private static void httpDeleteFromCloud(String url,
                                            String username,
                                            String password)
        throws Exception {
        DefaultHttpClient client = getHttpClient(username, password);
        HttpDelete httpDelete = new HttpDelete(url);
        client.execute(getTargetHost(url),
                       httpDelete,
                       getHttpContext());
    }

    // Helper method to run an command through an SSH session to the
    // HOST using the user name and private key provided. The method
    // returns the console print out on the remote SSH session.
    private static String runSsh(String host,
                                 String user,
                                 String privateKeyFileName,
                                 String command) throws Exception {

        report("    * Issue command:" + command + " to " + user + "@" + host);
        JSch jsch = new JSch();

        File file  = new File("config/" + privateKeyFileName);

        jsch.addIdentity(file.getAbsolutePath());

        Session session = jsch.getSession(user, host, 22);
        session.setUserInfo(new MyUserInfo());

        // To by pass public key verification of the newly created
        // host. In practice, the public key from the sshd on the
        // newly launched VM should be obtained through a secure
        // channel.
        java.util.Properties config = new java.util.Properties();
        config.put("StrictHostKeyChecking", "no");
        session.setConfig(config);

        session.connect();
        Channel channel = session.openChannel("exec");

        ((ChannelExec) channel).setCommand(command);
        channel.setInputStream(null);

        InputStream in = channel.getInputStream();

        ((ChannelExec)channel).setErrStream(System.err);
        channel.connect();

        String retString = new String();
        byte[] tmp=new byte[1024];
```

6

```

        while(true){
            while(in.available()>0){
                int i=in.read(tmp, 0, 1024);
                if(i<0)break;
                retString += new String(tmp, 0, i);
            }
            if(channel.isClosed()){
                break;
            }
            try{Thread.sleep(1000);}catch(Exception ee){}
        }

        channel.disconnect();
        session.disconnect();

        return retString.replaceAll("[\r]", "");
    }

    // Helper class to facilitate the authentication call back for
    // establishing the SSH session, required by JSCH library.
    public static class MyUserInfo implements UserInfo {

        public MyUserInfo() {
            super();
        }

        @Override
            public String getPassphrase() {
            return null;
        }

        @Override
            public String getPassword() {
            return  null;
        }

        @Override
            public boolean promptPassphrase(String arg0) {
            return false;
        }

        @Override
            public boolean promptPassword(String arg0) {
            return false;
        }

        @Override
            public boolean promptYesNo(String arg0) {
            return true;
        }

        @Override
            public void showMessage(String arg0) {
        }
    }
}
```

7

References

[Cockburn] *Writing Effective Use Cases*, Cockburn,Alistair, Addison-Wesley, 2001. Draft versions widely available on internet, e.g., http://www.infor.uva.es/~mlaguna/is1/materiales/BookDraft1.pdf.

[DMTF] "Use Cases and Interactions for Managing Clouds", Version 1.0.0, 2010-06-18, document number: DSP-IS0103, http://dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf; "Interoperable Clouds, A White Paper from the Open Cloud Standards Incubator", Version 1.0.0, 2009-11-11, document number: DSP-ISO101, http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf

[IETF-SCIM] Draft use cases for System for Cross-domain Identity Management (SCIM), under development within the IETF, http://tools.ietf.org/html/draft-zeltsan-scim-use-cases-00

[LIBCLOUD] *Apache "LibCloud"* is currently undergoing Incubation at the Apache Software Foundation, http://incubator.apache.org/libcloud/

[NSTIC] "National Strategy for Trusted Identities in Cyberspace, Creating Options for Enhanced Online Security and Privacy", June 25, 2010, http://www.dhs.gov/xlibrary/assets/ns_tic.pdf

[OGF] "Open Cloud Computing Interface - Use cases and requirements for a Cloud API", Jan. 2010, http://ogf.org/documents/GFD.162.pdf

[ORACLE] Oracle Cloud Business Support Use Cases, v0.4, http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/SAJACCTeleConf25/Oracle_Cloud_Business_Support_Use_Cases_v0.4.pdf

[SNIA] "Cloud Storage Use Cases", Storage Network Industry Association, Version 0.5 rev 0, June 8, 2009, http://www.snia.org/tech_activities/publicreview/CloudStorageUseCasesv0.5.pdf

[TMFORUM] Enabling End-to-End Cloud SLA Management, Version 0.4, September 2012, TR178, http://www.tmforum.org/TechnicalReports/TR178EnablingEndtoEnd/50148/article.html